Project no. 027087

TENCompetence

Building the European Network for Lifelong Competence Development

Project acronym: Integrated Project TENCompetence

Thematic Priority: 2.4.10

ID3.6 Architecture Design Document

Due date of deliverable: 01-01-2008
Actual submission date: 23-04-2008

Start date of project: 01-12-2005 Duration: 4 years

LOGICACMG

Revision 1.1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Building the European Network
For Lifelong Competence Development

TENCompetence IST-2005-027087

Building The European Network for Lifelong Competence Development

Project Internal Deliverable Report

ID3.6 Architecture Design Document

Work Package	WP 3 – Technical Design & Implementation of the Integrated System
Task	
Date of delivery	**Contractual**: 01-01-2008 **Actual**: 23-04-2008
Code name	ID3.6 **Revision**: 1.1 Draft ☐ Final ☒
Type of deliverable	Report
Security (distribution level)	Public
Contributors	Roy Cherian (University of Bolton), Arne Koesling (UHANN)
Authors (Partner)	Ruud Lemmers (LOGICACMG), Hubert Vogten (OUNL), Harrie Martens (OUNL), Phil Beauvoir (University of Bolton), Michele Dicerto (ILABS)
Contact Person	Ruud Lemmers (LOGICACMG)
WP/Task responsible	WP3 / Ruud Lemmers (LOGICACMG)
EC Project Officer	Mr. M. Májek
Abstract (for dissemination)	The document explains how the PCM server works and how to use version 1.0 of the PCM services to three target groups: 1. Designers and analysts who need to understand what functionality the PCM services offer. 2. Developers who will build new clients on top of the PCM services. 3. Developers who intend to extend and / or change existing PCM services. The document also describes the software architecture of the PCM server to developers that need to extend or change the PCM server. The views from the 4+1 approach by Kruchten are used to describe the software architecture. The description of the PCM services also forms ID3.7 – “Final API definitions for the second release, beta version”.
Keywords List	Software architecture, design, models, developer guidelines, PCM services

TENCompetence Project Coordination at: Open University of the Netherlands
Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands
Tel: +31 45 5762624 – Fax: +31 45 5762800

Table of contents

Version history

Version	Date	Description	Editor(s)
0.1	06-02-2008	Initial version.	Ruud Lemmers (LOGICACMG)
0.2	07-02-2008	Inserted deployment model, detailed PCM services, database model and the authorisation rules to create the first complete version.	Ruud Lemmers (LOGICACMG)
1.0	21-02-2008	Included review comments of Arne Koesling and Roy Cherian.	Roy Cherian (University of Bolton), Arne Koesling (UHANN) and Ruud Lemmers (LOGICACMG)
1.1	23-04-2008	Incorporated the comments from external reviewers.	Ruud Lemmers (LOGICACMG)

1. Introduction

1.1. Aim and background

The domain model and primary use cases (both found in [2] - TENCompetence Domain Model) were the basis for WP3 to develop a series of prototypes. Each prototype formed another step towards the creation of an initial integrated system. Version 1.0 of the PCM client and PCM server (both delivered as part of [7] - .2: First tested version of the Integrated TENCompetence System) form this initial integrated system.

Many aspects of the PCM client were described in [6] - .1 Architecture Design Document (its functionality), [8] - ID2.5 Elaborated Use Cases & Domain Model (its detailed processes) and WP9's user manual (its detailed functionality and how to use it, only available as a draft version). The documentation on the PCM server is more limited, especially when it comes to using the services it offers. These services will be the main basis for the clients from the aspect work packages and therefore require good documentation.

The first aim of this document is to provide the documentation required to understand and use version 1.0 of the PCM services. It does this by describing the services to three target groups:
1. Designers and analysts who need to understand what functionality the PCM services offer.
2. Developers who will build new clients on top of the PCM services.
3. Developers who intend to extend and / or change existing PCM services.

The description of these services also forms internal deliverable **ID3.7 – "Final API definitions for the second release, beta version"**.

The second aim of the document is to describe the software architecture of the PCM server to developers that need to extend or change the PCM server. The chapters of the document represent different views on the system, each putting their own piece of the puzzle in place.

The sections on the ideas and concepts underlying the services will help developers, designers and analysts understand what the system does and thus support both document aims.

1.2. Reading guide

Using the concepts from [1] - The 4+1 View Model of Architecture, and adding developer guidelines, the remainder of the document can be divided into five main parts:

1. **Logical view**: gives an overview of all services and explains the main ideas and concepts of the system.
2. **Process view**: shows the system's most important flows.
3. **Implementation view**: explains the main classes and describes all PCM services in detail.
4. **Deployment view**: provides a network topology view of the system, by showing the different TENCompetence nodes (machines) and their connections.
5. **Developer guidelines**: guidelines to help developers in writing code that is easier to integrate into the TENCompetence system. Guidelines are given for using Bugzilla (a bug tracking tool), unit testing, use of CVS (a system for version control of source code), coding conventions, developing user interfaces, and for development tools.

The use case view from [1] is not described in this document. Describing it would result in a set of technical use cases like “Manage Competence”, “Manage Competence Profile”, etc. Because there is no human user of the services this has little value. The client(s) used by a human user realizes a number of use cases by combining appropriate service calls. Documentation for the clients should describe their use cases.

Chapter 9 – “Recommended introductory reading” falls outside these five main parts. It provides a few links to documents that are very useful when getting started on work related to the TENCompetence project.

1.3. Acronyms

Acronym	Full description
API	Application Programming Interface
CCSI	CopperCore Service Integration
CDP	Competence Development Programme
CRUD	Create, Read, Update, Delete
CVS	Concurrent Versions System
EJB	Enterprise Java Bean
PCM	Personal Competence Manager
RCP	Rich Client Platform
SLeD	Service oriented Learning Design
WP	Work Package, one of the groups working in the TENCompetence project
WP3	Work Package 3, responsible for “Technical design and implementation of the integrated system”
WP5	Work Package 5, responsible for “Knowledge resources sharing and management”
WP6	Work Package 6, responsible for “Learning activities and units of learning”
WP7	Work Package 7, responsible for “Competence development programmes”
WP8	Work Package 8, responsible for “Networks for lifelong competence development”
WP9	Work Package 9, responsible for “Training”

2. Logical view

2.1. Introduction

This chapter explains the main concepts and ideas behind the server functionality. Each small section is dedicated to one specific concept.

2.2. Design class diagram

The PCM services have been derived from the functionality required by the rich PCM client (the RCP client built in Eclipse). The required communication is more fine grained than the standard "credit card check" example which becomes clear when depicting the PCM services in Figure **1. Design class diagram**. Future changes and additions to these services will be determined by the requests from the different work packages.

Each operation in one of the classes represents a PCM service. Many similar operations exist: create, retrieve, update, delete, share, register, complete and retrieveList occur often.

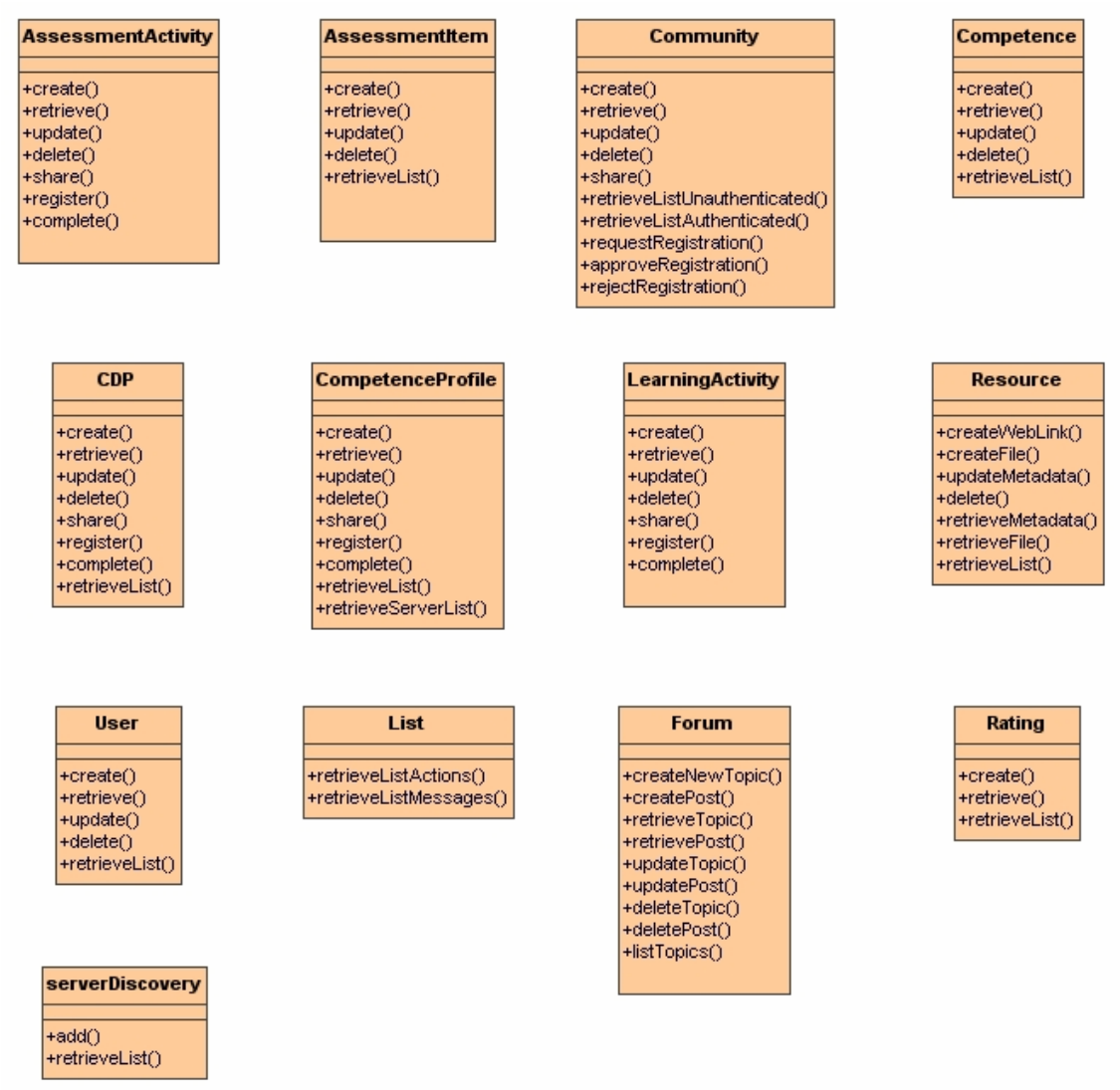


Figure 1. Design class diagram

2.3. CRUD operations

The standard CRUD operations form the bulk of the services. Each of these is used to manipulate single objects instead of a batch, e.g. “create Competence Profile Associate Professor”. There is no support yet for batch operations (e.g. “delete all Competence Profiles”).

Each of the entities has ‘create’ and ‘retrieve’ operations and almost all of them support ‘update’ and ‘delete’ as well. Deleting an object is only allowed when it is not in use by any other object to avoid inconsistencies in the relations between objects.

2.4. Authentication

Almost all services require users to identify themselves before being allowed to use the service. Identification is done by checking a username and password combination.

[Technical note: when setting up a new connection to the server, Tomcat checks the username/password using the digest authentication scheme.]

2.5. Authorisation

Users can choose which objects they want to share with others. Performing a certain operation on an object only succeeds when the user is allowed to perform that operation on that object. Also, when retrieving summary lists of objects only the objects the user is allowed to retrieve are returned.

When sharing an object, a user can choose from three options:
1. Private: don’t share this object. The object is only visible to the author.
2. Shared: the object can be retrieved by every user in the community. The author can change the object and can assign change rights to specific users.
3. Shared for update: the object can be retrieved and changed by every user in the community.

Please see *Section 8 Authorisation rules* for the detailed rules on authorisation.

2.6. Search

The search functionality is limited to retrieving a list of all objects related to a certain other object. In most cases this is of the form “list all objects of type X from community Y”.

> **Example**: list all Competence Profiles from Community “Psychology”.

2.7. Scope limitations for objects

All objects (except users) are limited by the boundaries of a Community. The objects are invisible within the context of other communities!

> **Example**: the competence “Communication skills” is created within the “Psychology” Community. When the same competence is needed in the “Project management” Community, a new “Communication Skills” competence needs to be created in this second Community.

Users are the only objects that can exist across multiple Communities. A user object is limited by a PCM server (a PCM server instance, to be precise). The same user object can be used in all communities of a PCM server installation, but it can't be used on another PCM server.

When needed, a person can create user accounts on multiple PCM servers. The drawback is that a person will have multiple user accounts. The PCM client manages the different user accounts to make it as easy as possible for a person to use multiple PCM servers.

2.8. Personal data

By registering for Competence Profiles, CDP's and Learning Activities, users can keep track of what they are using. "Completing" those objects keeps the record of what they accomplished up to date. Using these flags is important to provide supporting statistics to other users (e.g. "25 users are using the 'Informal plan to become an Associate Professor' ").

Completion and registration status for objects are part of a user's "personal data". The other personal data items are the accomplished competence levels and the created flag (indicating a user created a certain object).

2.9. Mapping the Domain model to the PCM services

Domain model	**Class diagram**	**Explanation**
Assessment activity	AssessmentActivity + AssessmentItem	AssessmentActivities include one or more AssessmentItems.
Learning network	Community	
Competence + result	Competence	When a competence is completed at a certain level, this result is stored by the Competence service.
Competence development programme + result	CDP	Like Competence.
Function / job in domain + result	CompetenceProfile	Like Competence.
Activity + result	LearningActivity	Like Competence.
Knowledge resource	Resource	
Actor	User	The role of a person is determined by the context, there is no fixed role. E.g. user Peter can both create new communities (= an administrator role), create new CDP's (= a teaching or course design role) and follow a CDP in another community (= a learner role).

Domain model	Class diagram	Explanation
Activity + notification service	List	Although named "retrieveListActions", it returns a list of activities. The "retrieveListMessages" operation is the first part for a notification service.
Communication & collaboration facilities	Forum	The forum provides functionality to create threads & posts. Chat functionality is available in the infrastructure as a third party product (and therefore not depicted in the class diagram).
Rating & comments	Rating	
-	serverDiscovery	This provides functionality to find (discover) available PCM servers. There is no counterpart in the domain model.

2.10. Mapping the design class diagram to the physical services

It's easy to map the logical services to the physical services by using the class names from the design class diagram and the section names from section *2.2 Design class diagram*. Section *7 PCM services reference* explains in detail how to use the different operations.

3. Process view

3.1. Introduction

This chapter highlights a few aspects of the actual implementation of the system. It starts with two different views on the main process: processing service requests. Knowing this flow will help in understanding the physical package diagram that follows. The flow and the package diagram form a good basis for understanding the PCM server.

The details of the services also belong to the implementation view. Because of their length, they are not part of this chapter but have been placed in Section *7 PCM services reference*.

3.2. Processing service requests

Using two different diagrams, a sequence diagram and a collaboration diagram, the main process of the system is illustrated. The sequence diagram focuses on the different layers of the system by using the layers as its swimlanes.

3.2.1. Sequence diagram

Figure 2 shows the steps in handling a request and the involvement of the different application layers.

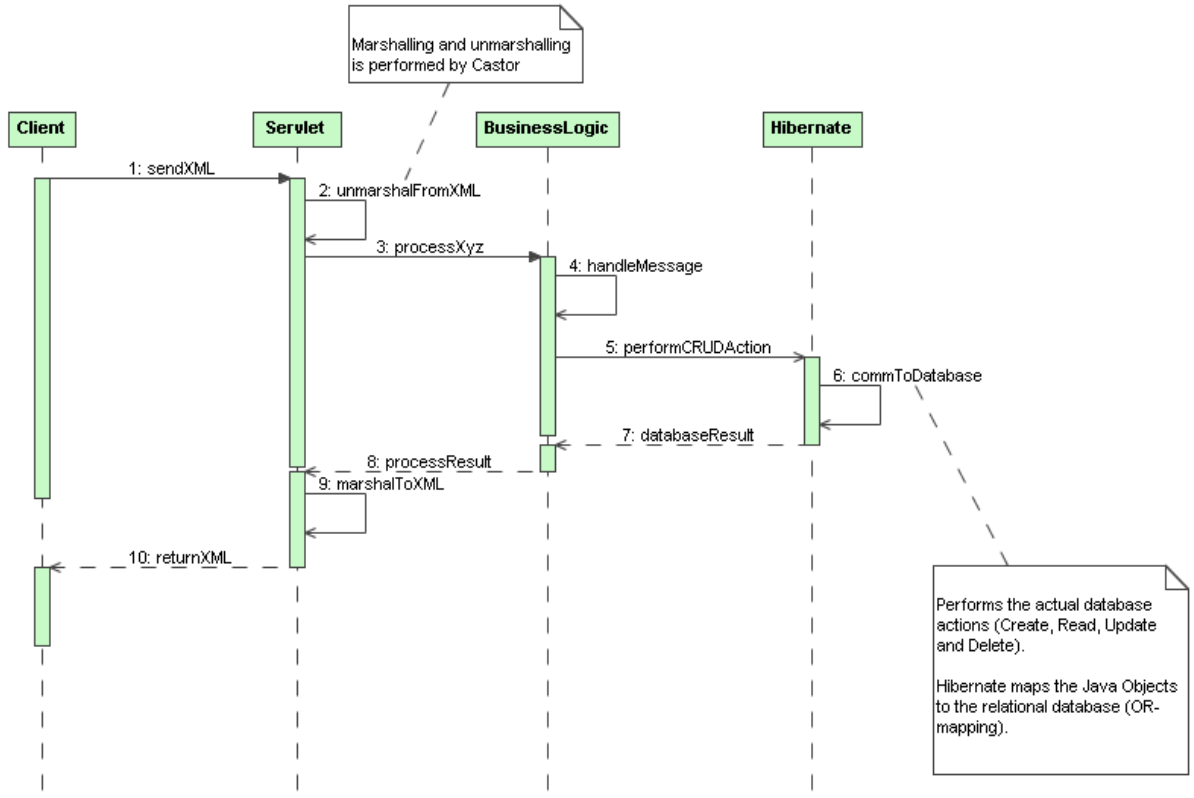


Figure 2: Process service request - sequence diagram

- Client: sends and receives XML messages.
- Presentation layer: a servlet receives an XML message and converts this to a request to the Business Logic layer.
- Business Logic layer: guides the "actual" work (consistency checks, required persistence operations, algorithms, …).
- Data layer: the Hibernate component handles the database transactions.

3.2.2. Collaboration diagram

Figure 3 shows the most important classes of the PCM server and their role.

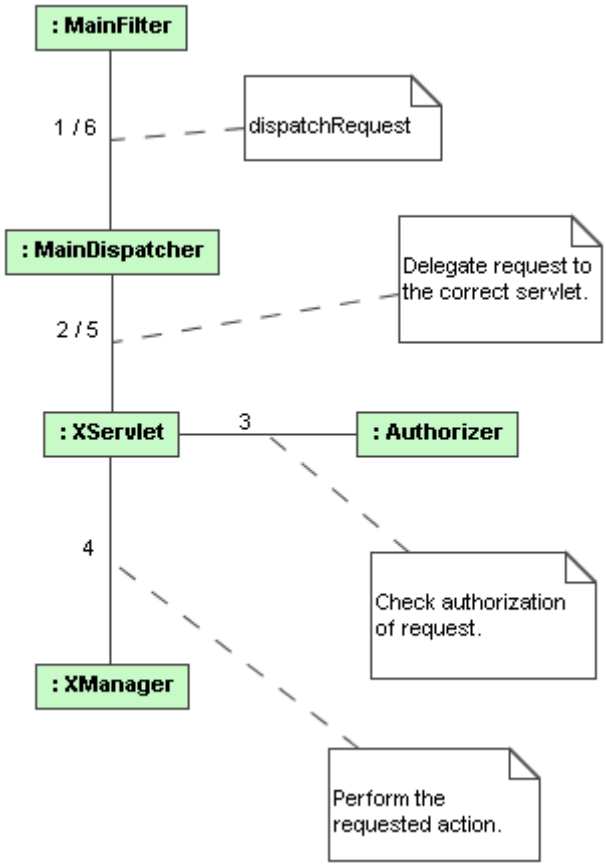


Figure 3: Process service request - collaboration diagram

1 MainFilter performs general actions required for each request: determines the requesting user, starts a new database transaction and then passes control to the MainDispatcher.
2 MainDispatcher parses the URI of the request to determine the servlet for handling this request and passes control to the appropriate servlet.
3 XServlet is the entry point for receiving and returning XML messages for entity X (each entity has its own servlet class, e.g. CompetenceProfileServlet, CompetenceServlet, etc.). It uses Authorizer to check if the user is allowed to perform the requested action on the provided data.
4 When the request is authorized, XServlet lets the XManager do the actual work for the request. XManager performs the business logic for entity X (each entity has its own manager class, e.g. CompetenceProfileManager, CompetenceManager, ...).
5 XServlet gives control back to the MainDispatcher after completing the request.
6 MainDispatcher gives control back to the MainFilter.
7 MainFilter ends the transaction and returns the result to the client.

3.3. Notification messages flow

The notification messages are, in version 1.0, only used for events related to people registering for communities. From a functional point they are not important, but they caused a lot of confusion during development. To prevent future confusion, this section uses an activity diagram to show which messages are sent, when they are sent and to which recipients they are sent.

Note: “sending” can be misleading in this context as the server only replies to client requests. To receive messages, the client needs to send a request for messages to the server using the List.retrieveListMessages service.

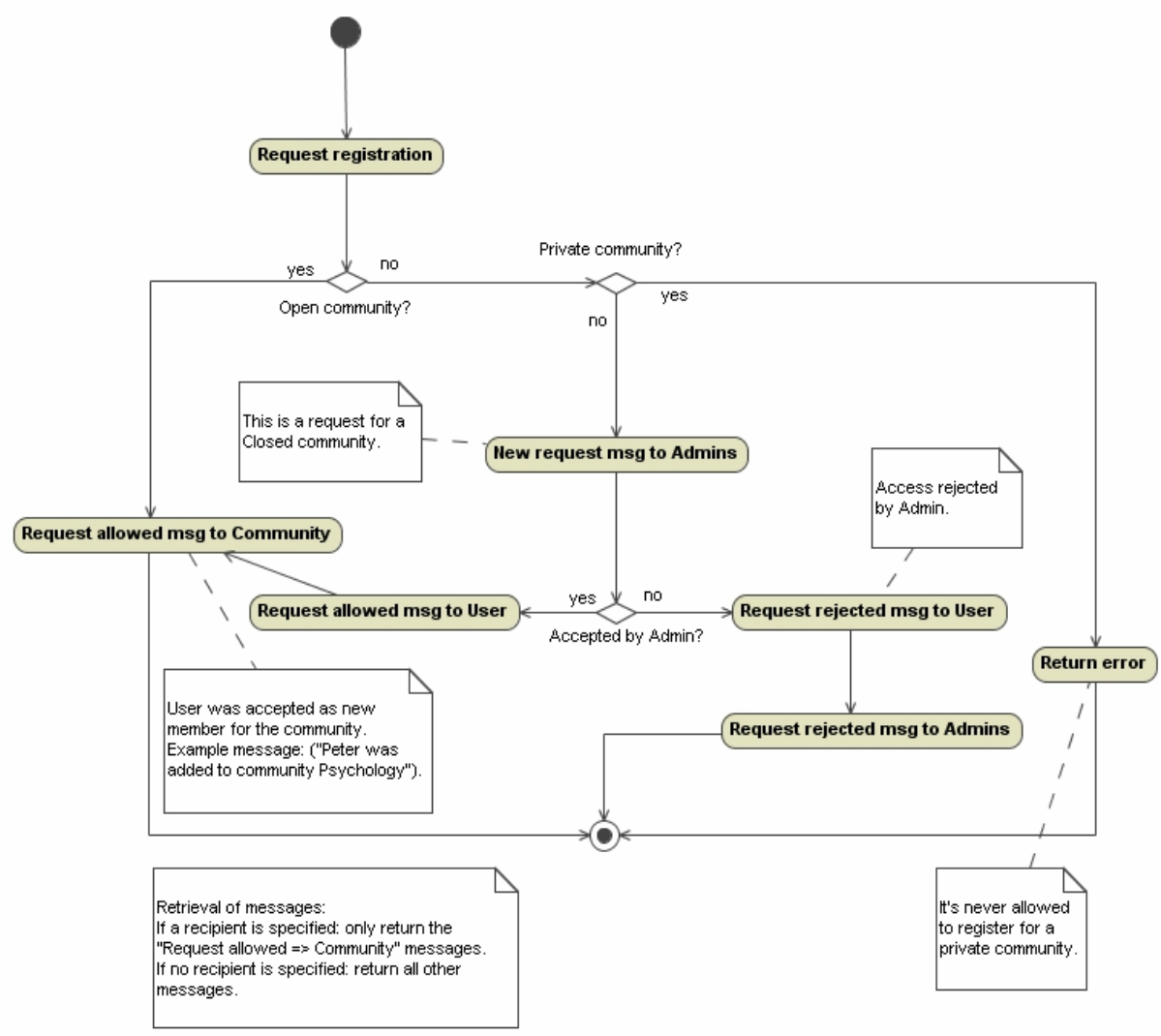


Figure 4 Activity Diagram - Notification Messages

An example of a path through this flow, for a user Peter trying to register for the Psychology community, which is a closed community (to which an admin user needs to grant access):

1. User requests registration for the Psychology community.
2. Open community? No.
3. Private community? No.
4. A “new request” message is created for all admin users of the Psychology users. When the client of an admin user requests the messages, this “new request” message will be amongst the returned messages. The text in the message will be similar to “User Peter is requesting access to community Psychology”.
5. Accepted by Admin? Yes.
   An admin user accepts Peter as a new user for the Psychology community.
6. A “Request allowed” message is sent to Peter. The text in the message will be similar to “Welcome to the Psychology community”.
7. A “Request allowed” message is sent to all members of the Psychology community. The text in this message will be similar to “User Peter has been added to the Psychology community”.

4. Implementation view

4.1. Package diagram

Figure 5: Package diagram shows the package structure as it is physically implemented in the code.

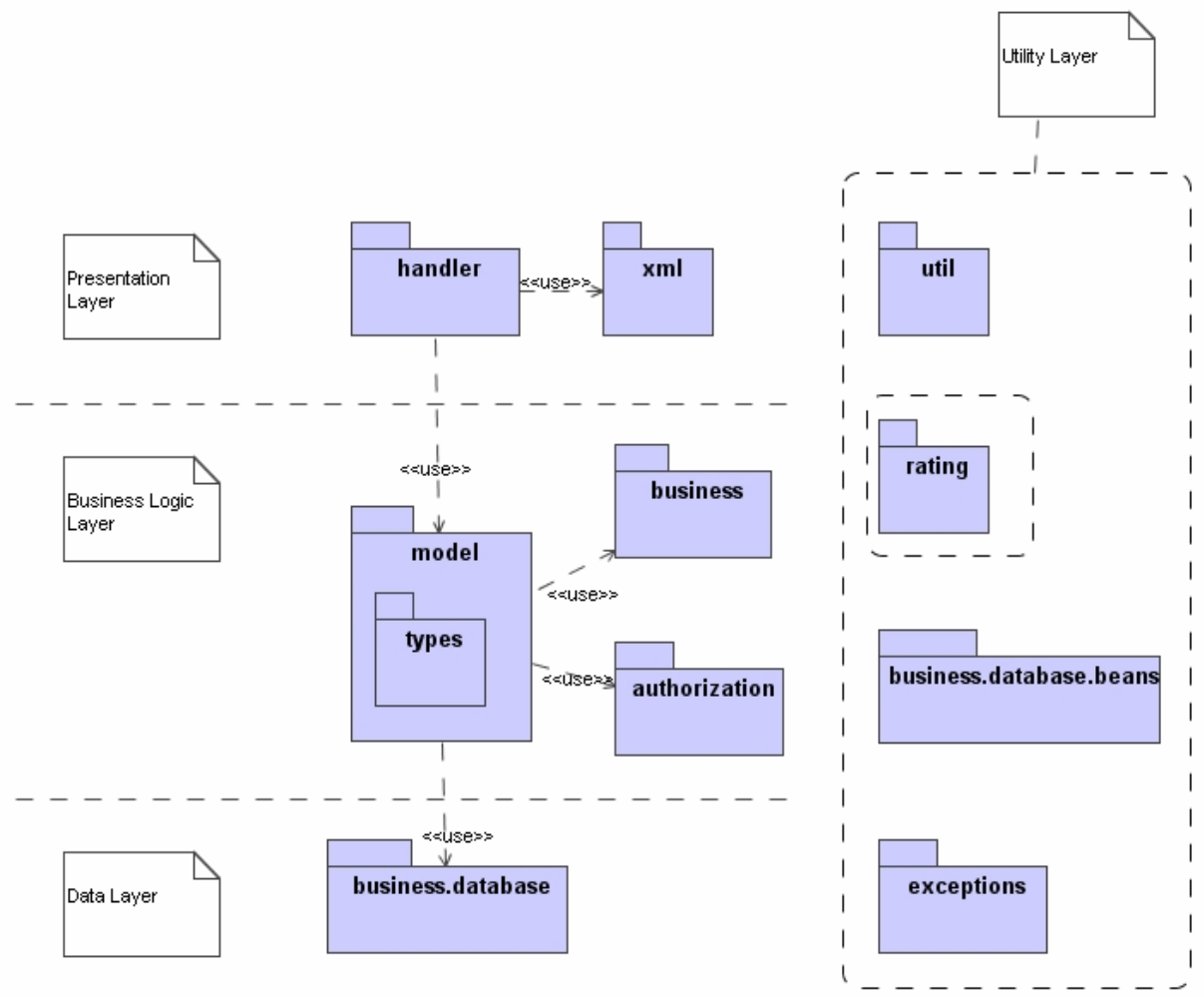


Figure 5: Package diagram

Package	Description
xml	The data object classes used by Castor (an XML marshaller / unmarshaller) for converting XML messages to Java objects (and vice versa).
handler	Contains the MainDispatcher and MainFilter control classes (as mentioned in section *3.2 Processing service requests*) and the servlets for receiving and returning XML messages (CompetenceProfileServlet, CompetenceServlet, ...).
authorization	Checks whether a request should be allowed or rejected, by using the Authorizer class.
model	Contains the main business logic classes, contained in CompetenceProfileManager, CompetenceManager, …

Package	Description
types	Contains a number of common reference types: • GroupType: Admin, Member, User. • MessageType: Request, RequestAllowed, RequestDenied. • ObjectType: Action, Assessment, Community, … • SharedType: NotShared, Shared, SharedForUpdate. • StatusType: Completed, NotCompleted. • SubscriptionType: NotRegistered, Pending, Registered, Rejected. • VisibilityType: Closed, Open, Private.
business	Helper classes for the Dozer tool (a JavaBean mapper), which automatically converts from "Java XML" (used by Castor) to "Java Database" (used by Hibernate) objects. There are two separate formats, because the information and structure of XML messages differs significantly from the information and structure in the database.
business.database	Generic classes for persisting data to the database and retrieving data from the database. Note: physically this package is placed within the "business" package. It is shown differently in the picture to display where it conceptually lies. Physically splitting the package up would be a small architectural improvement.
Util	General utility classes (in version 1.0 there's exactly one class – PropertyLoader – in it), which can be used in all layers.
Rating	All classes required for ratings: receiving / messages, data access object and database storage / retrieval. Because these classes cover Presentation, Business Logic and Data, the package has been placed within the Utility Layer.
business.database.beans	The data object classes used by Hibernate to persist Java objects to the relational database and to retrieve objects from that database. The package is placed in the Utility Layer because these data classes are also used for transporting/using data in the Business Logic Layer. Note: physically this package is placed within the "business.database" package. It is shown differently in the picture to display where it conceptually lies.
exceptions	The custom exception classes for TENCompetence exceptions. These exception classes can be used in all layers.

4.2. Database model

4.2.1. Introduction

The TENCompetence domain model describes, among other things, a set of entities and their relationships as used within the TENCompetence system. A consequent requirement is that the TENCompetence System needs to store these entities.

The first issue concerns the mapping of the domain model's entities and relationships to a data model respecting its needs. There exists, where possible, a direct mapping from the domain model to the data model. This means that there should be a data structure for each entity; one for a Community, one for a Competence and so on. The same applies for relationships.

The data model should also provide some additional data structures to support the client's operation and requirements such as "Permission", "Users", "Ratings", "Agent Messages" and "Forum".

4.2.2. Overview

To help in getting a view of the overall structure of the database model, it is split up into seven logical sections:

1. Admin - Administration tables.
2. Lookup - Support tables used to locate the information.
3. Forum - Tables used for the forum service.
4. Personal - Tables used to store information directly related to a specific user
5. Canonical - Tables deriving directly from the domain model.
6. Agent Messages - Table used by the agent for messages
7. Rating – Tables used to store objects rating.

One table in the datamodel can't be placed in a specific section. This is the *object_type* table which is a reference table used by multiple sections.

The Hibernate product ([23])is used for object/relational persistence and query services. It enables development of persistent classes following the object oriented idiom - including association, inheritance, polymorphism, composition, and collections.

4.2.3. Table description

Admin, Lookup

Table Name	Description
Admin section	
Admin	The admin table contains the object's id and the related owner.
Lookup section	
lookup_table	This table is used to create a relation between external id (coming from the client) and internal unique id. It is also used to speed up the search. The field object_id give the possibility to concentrate the search only on the correct object table.

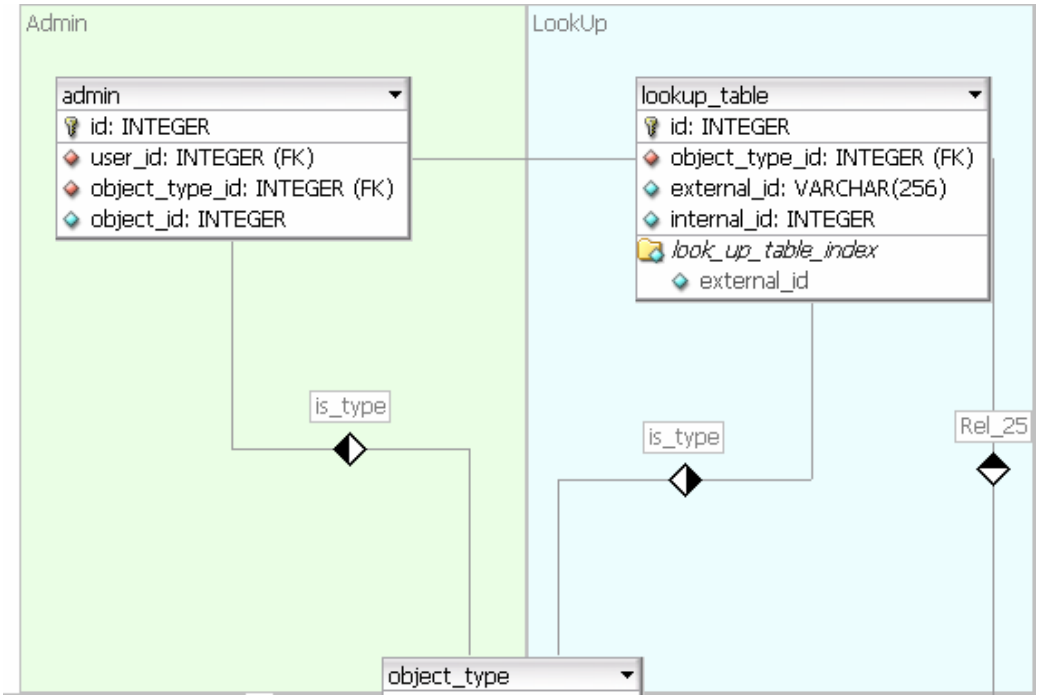


Figure 6: Admin, Lookup

Forum

Table Name	Description
Forum Section	
Topic	Represents a topic within a forum. It stores the id of the creator, the title, the creation timestamp.
Post	Represents a post in a forum. Through the relation parent_post_id, it is possible to recreate the structure of the messages within a topic.

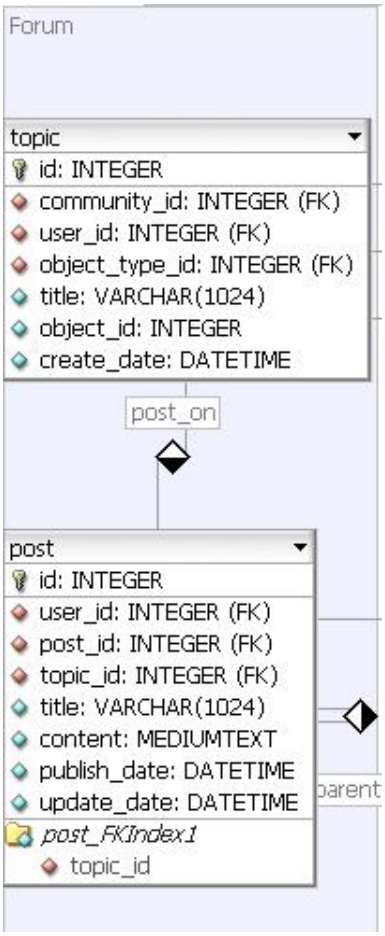


Figure 7: Forum

Personal

Table Name	Description
Personal Section	
personal_data	It represents the data strictly related to a user like a subscription to a community, the status of a particular action and so on.
Subscription	It represents the status of a subscription. Possible values are: • notregistered • registered • pending • rejected
Status	It represents the status used within personal_data. Possible values are: • true (Completed) • false (Not Completed)

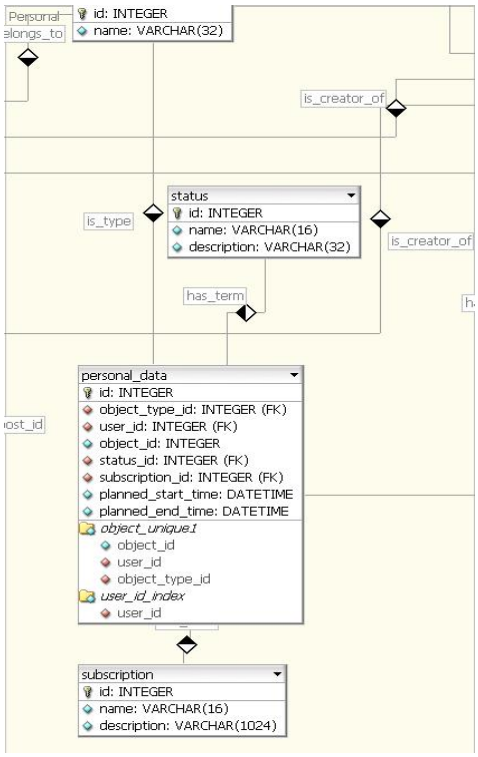


Figure 8 Personal

Canonical

Table Name	Description
Tables deriving from domain model	
action	Contains Action data (to store Activity and Route data, which are specialized versions of an action).
action item	It describe the item composing the action (resources)
cdp	It represents a competence development plan.
cdp_item	A competence development plan may contain ordered actions. This information is stored here. The order is maintained by sort_key field
user	It contains usual data of a user, like user name, display name, password.
user_role	It contains the role of an user
resource	All information related to a resource is stored here. Fields are very simple to understand. A particular mention for licence that describe the licence (free, commercial etc)
competence	This table describe a competence.
user_competence_level	This table contains the level on a competence for an user. It relates user table with competence table.
competence_profile	It describes, with cp_item and folder, a competence profile. It contains also information on the creator (a user) and if the competence profile is shared with other users.
cp_item	A competence profile can be composed by multiple competences. This table create the needed structure to relate a competence with a competence profile.
folder	Used create groups of competences within a competence profile.
assessmentitem	Used to relate questions to an action.
item	Represents a question and its answers (right + wrong answers)
community	It describes a community.

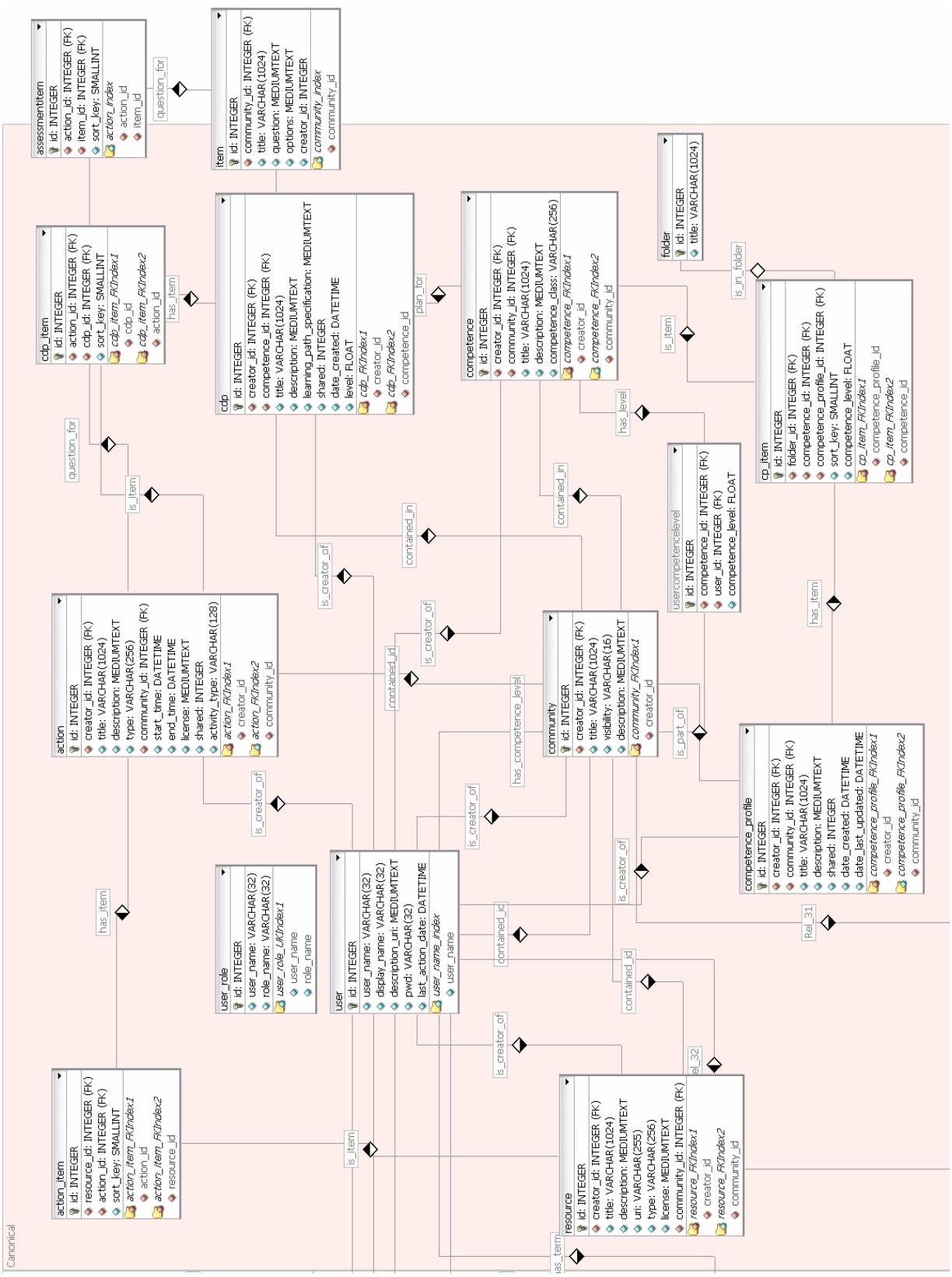


Figure 9 Canonical

Rating / Agent Messages

Table Name	Description
Rating Section	
Ratings	This table store the rating on a particular URI. It also includes the author, comment on rating.
accumulated_rating	It stores the sum of all ratings for an object. Used to calculate the average rating.
Agent Messages Section	
Message	Contains the messages sent from the agent to a user.

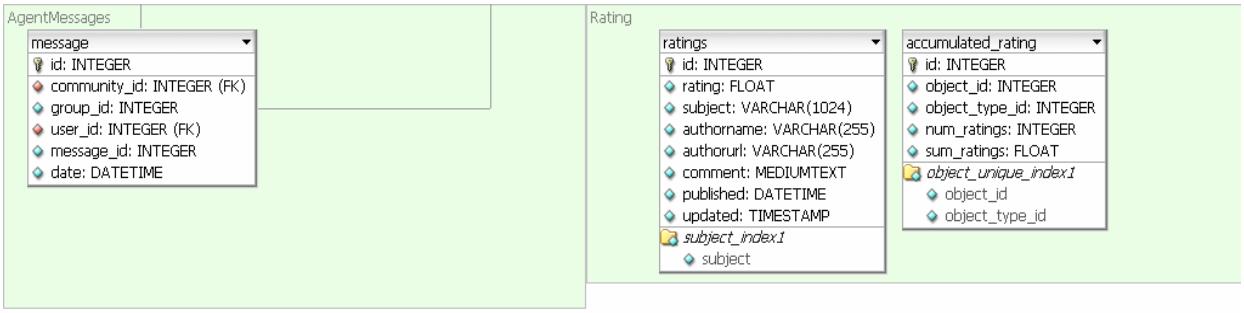


Figure 10 Ratings, Agent Messages

Others

Table Name	Description
Others	
Object_type	This table is a simple reference table used to list various object types used in the Data Model. Possible values of name are: 1. Action 2. Resource 3. Community 4. Competence 5. CompetenceProfile 6. CompetenceDevelopmentPlan 7. User 8. Topic 9. Post 10. Assessment 11. Item

4.3. PCM services

The API for PCM services are modelled using the REpresentational State Transfer (REST) approach. See http://en.wikipedia.org/wiki/Representational_State_Transfer for a detailed explanation of REST.

The following summarises salient features of REST based services on PCM:

1. Each "CRUD" operation (Create / Read / Update / Delete of an object) is matched to an HTTP method, according to the table below.

CRUD operation	REST (HTTP method name)
Create	POST
Read	GET
Update	PUT
Delete	DELETE

2. Every object is regarded as a resource and has its own unique URI. The HTTP method determines what action is performed on the resource (create it, read it, …). The type of each entity in URIs can be recognized by their extension.
3. The clients have the responsibility to generate identifiers and creating URI's. e.g. {community-guid} stands for the identifier of a certain community. When trying to create an object with an identifier that's already in use, the server will return an error message.

Example: sending a POST message to URI http://pcm.mysite.com/245897815.community/78245-bd736.assessment will try to create an assessment with identifier 78245-bd736 within the existing community 245897815.

Most of the documents passed between server and client contain three distinctive sections. These are:

- A canonical part wrapped by the <canonical> tag. The canonical part describes the part of the information for an entity that is generic for all users.
  Note: in some documents personalized information is mixed into the canonical part when performing a GET operation for performance. This personalized information is never passed back when doing a PUT or POST operation!
- A personal part wrapped by the <personal> tag. This personal part contains the personalized aspects of the entity for the user passing the request.

⇨ A rights part wrapped by the <rights> tag. This rights part contains all profile information regarding access rights for the entity.
Note: passing this information in a PUT or POST operation is only allowed whenever the user is also the owner of the entity. So the owner is the only user that has the right to change rights.

Details of the APIs used for PCM services are given in Section 7 (PCM services reference).

4.3.1. Examples

Below are two examples that illustrate how the PCM services can be linked together. "Using a Competence Profile" shows how to find and start using a certain Competence Profile. "Add an activity" creates a personal CDP by adding an activity to an existing CDP. They use a short scenario to illustrate which service calls to use to complete these tasks.

Using a Competence Profile

1. User.create
   Creates a new user.
2. CompetenceProfile.retrieveAnonymousList.
   Retrieve a summary list of all shared competence profiles on the whole server.
3. Community.requestRegistration
   Requests registration for the "Psychology" community. Registration is required before detailed objects from this community can be retrieved. We assume this is an open community, which results in automatic approval of the registration request.
4. CompetenceProfile.retrieve
   Retrieve the"Associate professor of Psychology" competence profile, to check if it's an interesting competence profile.
5. CompetenceProfile.register
   Registers the user for competence profile "Associate professor of Psychology".
6. Etc.

Add an activity

1. CompetenceProfile.retrieve
   Retrieves the "Associate professor of Psychology" competence profile. This assumes the person already has a user account and is registered for this competence profile.
2. CDP.retrieve
   Retrieves the CDP linked to the "Communication skills" competence for the current user. This assumes the user linked a CDP to this competence before.
   The user views the learning activities and decides one should be added to the list.
3. CDP.create
   The user decides to create his personal CDP, instead of extending the existing one. Creating a CDP automatically registers the user for this CDP.
4. List.retrieveListActions
   Retrieves all Learning Activities and Assessment Activities from the Psychology community, enabling the user to choose which activity (or activities) to add to the plan.
5. CDP.update
   The user picks the "Debating competition" activity and adds it to the CDP by sending an update.

5. Deployment view

The nodes required to run the full functionality of TENCompetence are depicted in Figure 11 TENCompetence Deployment diagram. The figure shows two layers. The first layer represents the client software. In the case of the diagram this is the Personal Competence Manager. This client can be deployed on a computer running either Microsoft Windows, Linux or Mac OS/X.

The second layer consists of TENCompetence server nodes. The first node is the TENCompetence Application Server. This server hosts a Tomcat servlet container. Tomcat runs the discovery service containing a list of discoverable TENCompetence server deployments and the core components of the TENCompetence server implementation. The discovery server may be omitted or alternatively be installed on a separate application server. On the same application server an instance of OpenFire must be installed to provide chat functionality to the connected clients. The application server also hosts a MySQL database management system. This database is used by Tomcat and OpenFire for storing and retrieving the authorization information. Furthermore the TENCore uses the database for its persistence. Additionally a CopperCore Application Server can be installed next to the TENCompetence Application server. The CopperCore Application Server hosts a JBoss EJB container on which the SLeD frontend provides access to the TENCompetence version of the CopperCore IMS Learning Design engine ([5]). This CopperCore version is created by Work Package 6.

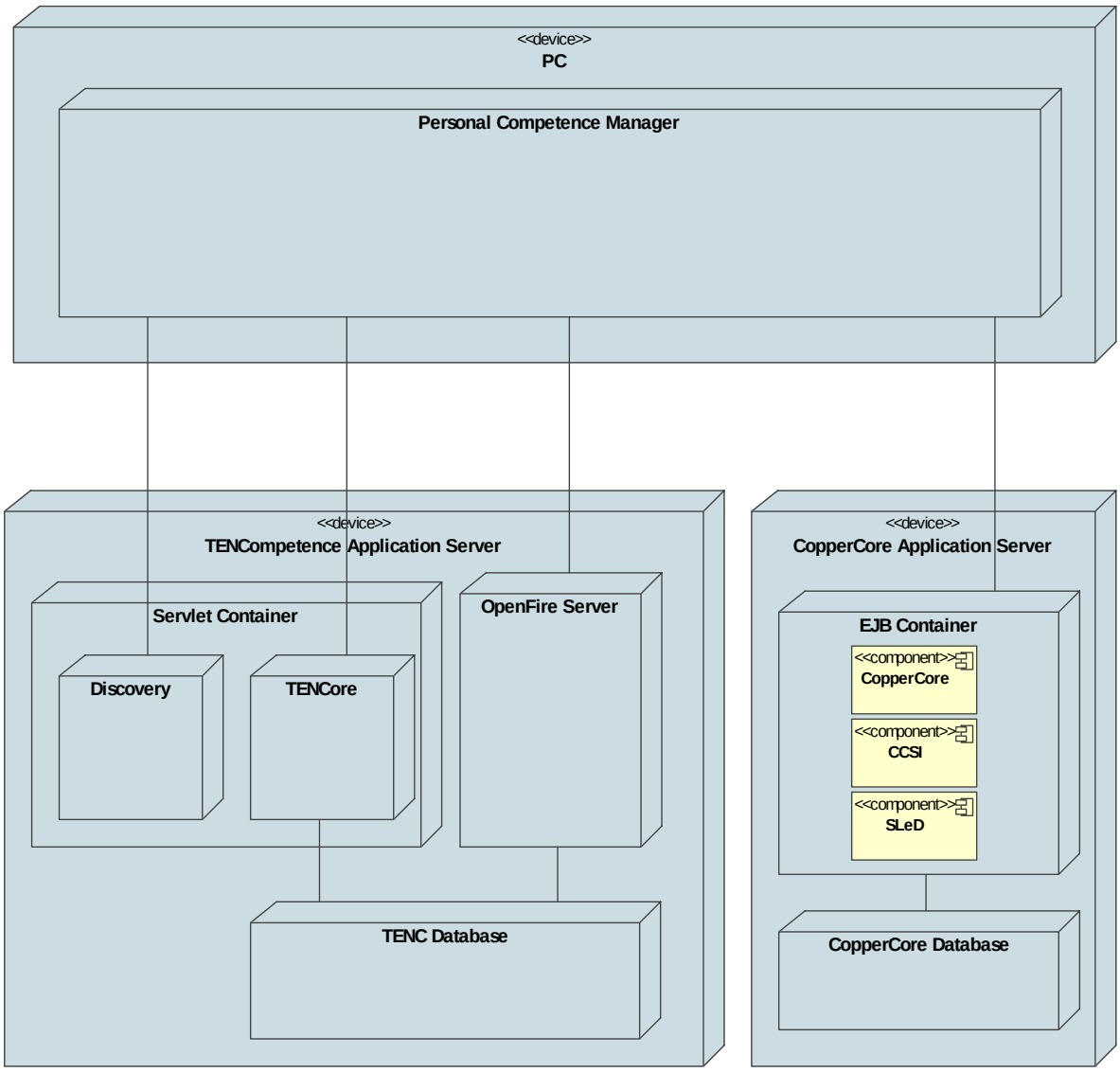


Figure 11 TENCompetence Deployment diagram

6. Developer Guidelines

6.1. SourceForge and Bugzilla

6.1.1. Background

Development work in the TENCompetence project requires the use of a server-based solution in order to track bugs, feature requests, and enhancements. The project is already using SourceForge ([10]) as a CVS repository and point of presence. The SourceForge hosting facility includes a bug tracking service. However, the TENCompetence project requires a more substantial and fully featured bug tracking service therefore an externally hosted Bugzilla Service is employed, located at Wush ([11]).

6.1.2. Bugzilla

Bugzilla is a "defect tracking" system that can track bugs, feature requests, and patches. In spite of its moniker, it used for more than just bug tracking. An example of a "bug" could be an item that tracks the progress of a feature request or enhancement. Bugzilla is free and customisable but has to be set up and hosted on a server. Bugzilla is used by many well known projects including the Mozilla Project, Eclipse, and NASA.

Features included can be found at http://www.bugzilla.org/features/

An example of a Bugzilla bug report page can be found here - http://wush.net/bugzilla/rulem/show_bug.cgi?id=125

Bugs and issues can be submitted by anybody, and will be assigned to a particular developer after a process of triage. Various status updates for each bug are allowed, together with user notes and bug examples.

In order for a bug, feature enhancement or issue to be addressed properly, the bug needs to be described clearly with steps to reproduce, and example screen shots or error messages if appropriate. Once assigned to a developer or developers the bug can be assessed and a fix submitted to the CVS repository (if possible) with the bug being marked as "Fixed". Further testing of the fix should be done by other testers until satisfactorily proven to be resolved when the bug can be marked as "Closed". Naturally, not all bugs or issues can be resolved or fixed and these should be marked appropriately in the Bugzilla report.

6.2. Unit Testing and JUnit

6.2.1. Background

Testing of the TENCompetence system software components is an integral part of the development process

JUnit is a unit testing framework written for the developer who implements unit tests in Java. It is free, Open Source and is easily integrated into the developer's test environment as either a standalone library or in tight integration with the Eclipse IDE.

The junit.jar library jar is linked to the code containing the tests' classpath. JUnit is currently at version 4.4.x, the previous popular version being 3.8.x. Eclipse supports and ships with both versions. JUnit ships with many useful features:

- A variety of front-ends that can display the results of the tests – command line, AWT, Swing
- Separate classloaders for each unit test
- Integration with popular tools like Ant, Maven and Eclipse

6.2.2. Integration with Eclipse

The development IDE and target platform chosen for the project is Eclipse. The Eclipse IDE workbench ships with a built in JUnit testing framework that is fully integrated into the Java development process. This makes it extremely convenient for developers to test their code instantly with graphical feedback in the JUnit View or by running test suites.

Eclipse contains the following JUnit integration features:

- JUnit View provides instant feedback of passing and failing tests
- Ability to run single or multiple tests from Eclipse
- JUnit functions available from contextual menus

6.2.3. JUnit 4

The default version of JUnit used in Eclipse is version 3.8.x. This version works with Java versions 1.4.x and above. The current version of JUnit is 4.4.x but depends on features provided by Java version 5.0 and above in order to compile and run.

The main features of JUnit 4 are

- It Requires JDK 5 to run
- Test classes do not have to extend from junit.framework.TestCase
- Test methods do not have to be prefixed with 'test'
- There is no difference between the old assert methods and the new assert methods
- Use @Test annotations to mark a method as a test case
- @Before and @After annotations take care of set up and tear down
- @BeforeClass and @AfterClass annotations take care of one time set up and one time tear down
- @Test annotations can take a parameter for timeout. Test fails if the test takes more time to execute

- @Test annotations can take a parameter that declares the type of exception to be thrown
- JUnit4Adapter enables running the new JUnit4 tests using the old JUnit runners
- Old JUnit tests can be run in the new JUnit4 runner

It is best to use the latest version of JUnit unless there are good reasons not to do so.

6.2.4. JUnit TENCompetence source folder structure and naming conventions

JUnit tests should be located in a separate module that maintains a mirror package structure of the original module that is being tested. This ensures that the main module does not have to link to or contain the junit.jar library, and also so that users can check out the code from CVS without having to get the tests as well, these remaining optional in the corresponding JUnit tests module. This is the practice that the Eclipse development team use themselves.

For example, a plug-in module called `org.tencompetence.client` would have its tests in a separate module called `org.tencompetence.client.tests` and the package structure would be as follows:

The original module:

```
org.tencompetence.client
      |----src
            |----org.tencompetence.client
                              |----package1
                                        |----StringUtils.java
                              |----package2
                              |----package3
```

And the JUnit test module structure:

```
org.tencompetence.client.tests
      |----src
            |----org.tencompetence.client
                              |----package1
                                        |----StringUtilsTest.java
                              |----package2
                              |----package3
```

File names of tests should follow the naming convention of “Class to be tested” followed by the suffix “Test”. For example:

Class to be tested:	StringUtils
Name of test class:	StringUtilsTest

6.2.5. Automated and Manual Testing frameworks

A developer can run single or multiple tests from within Eclipse “on the fly” as needed. Additionally, we will be running nightly builds and automated scripts that will run all JUnit tests and report the results as HTML files. We have already created an Ant script to perform the task of running automated JUnit tests.

6.2.6. Guidance for writing tests

Providing best practice guidance for writing JUnit tests is beyond the scope of this document. However, developers are recommended to study the materials mentioned in reference[13-16].

6.3. The Use of a CVS Repository in the TENCompetence Project

6.3.1. Background

The TENCompetence development process works in an open manner, utilising an Open Source licensing model and attempting to maintain transparency at all levels. One means to accomplish this transparency is to ensure that all code, documentation and development artefacts are available in a **Concurrent Versions System** (**CVS**) **Repository**.

6.3.2. CVS and SourceForge

The TENCompetence project uses SourceForge for its CVS repository. This is a free service that provides hosting for Open Source projects and provides additional services such as issue tracking, forums, project presence, CVS and Subversion Repositories.

6.3.3. Access rights and user permissions

We have allocated key project staff as CVS administrators and assigned key developers with CVS developer status. By default, CVS access is read only, and this includes public anonymous access. However, developers are granted write access on a per-module basis

In order for a developer to be granted write access to files in a module or modules, the following steps have to be taken:

1. The aspiring developer has to register at SourceForge with a user name.
2. A SourceForge administrator has to add the user as a developer to the SourceForge project admin page.
3. A SourceForge administrator has to edit the `avail` file to include the user name and module name(s) required for read/write access.

As of the time of writing of this document, the `avail` file is as follows:

```
unavail
avail|hubertvogten,harriemartens,rulem,phillipus,scottwilson,ps3com
avail|rp_cherian|org.tencompetence.client
avail|rp_cherian|org.tencompetence.client.comms.chat
avail|rp_cherian|org.tencompetence.client.comms.skype
avail|arnewolf2|org.tencompetence.tencc.questionauthoring
avail|dicerto,r_celle,sheyenrath,phensgens|org.tencompetence.tencs
avail|dicerto|org.tencompetence.tencs.coordination
avail|dicerto|org.tencompetence.tencs.policy
avail|krassen,slavi_marinov,natanasov,elen_de,dicerto,academika,carlo
s_mendez,gzenz,sergejzerr|wp5
avail|academika,nilocnel|wp6
avail|kaerger|wp7
      avail|krassen,patzzzo,heitara,mirriam_koteto,aldi_12346,davorme
      ersman,triathlon98,stanley360|wp8
```

The first line, “unavail”, ensures that by default all developers do not have write access to the repository. The line “avail” is followed by user name(s) and the name of the module that is given write access. Full documentation for editing the avail file is available at the SourceForge website.

6.3.4. Integration with Eclipse

The development IDE and target platform chosen for the project is Eclipse. The Eclipse IDE workbench ships with a built in CVS client which is fully integrated into the Java development process. This makes it extremely convenient for developers to synchronize their workspace with the CVS repository and to monitor history, annotations and file comparisons within the IDE. Thus, no other CVS client is needed to perform all common CVS tasks.

6.3.5. Repository details

The SourceForge CVS connection details are as follows:

Web Page:	http://sourceforge.net/cvs/?group_id=159487
Host location:	tencompetence.cvs.sourceforge.net
Repository Path:	/cvsroot/tencompetence
Connection type:	extssh
Public User Name:	anonymous

6.3.6. Modules and folder structure

All code is deposited under the “HEAD” CVS location. Work Package 3 code is placed at the top level under HEAD, while other Work Package code is placed in appropriately named subfolders (“wp6”, “wp7”, “wp8”, and so on).

The naming convention for a code module for the TenCompetence client application is the project namespace (`org.tencompetence`) followed by the module name, as follows:

```
org.tencompetence.modulename
```

Where “modulename” is an appropriate name of the module. For example, “org.tencompetence.client”

As an example, if one had a module named "org.tencompetence.myproject" that came under the remit of WP5 then a developer using Eclipse would perform the following steps:

1. Right-click on the Project in the Package Explorer in Eclipse
2. Choose Team->Share Project...
3. Select the 10Competence Repository
4. Choose "Use specified module name" which would be `wp5/org.tencompetence.myproject`

6.3.7. JUnit folder structure

JUnit tests relating to a module should be organised in a separate module that maintains a mirror package structure of the original module. This ensures that the main module does not have to link to or contain the junit.jar library, and also so that users can check out the code without having to get the tests as well, these remaining optional in the corresponding JUnit tests module. This is the practice that the Eclipse development team use themselves.

For example, a plug-in module called `org.tencompetence.client` would have its tests in a separate module called `org.tencompetence.client.tests` and the package structure would be as follows:

The original module:

```
org.tencompetence.client
|----src
|----org.tencompetence.client
|----package1
|----package2
|----package3
```

And the JUnit test module structure:

```
org.tencompetence.client.tests
|----src
|----org.tencompetence.client
|----package1
|----package2
|----package3
```

For more information on JUnit and package structures, see the section on Unit testing.

6.3.8. Other considerations when using folders

The "bin" folder and any folders that contain compiled code files (*.class files for Java) should be excluded from the CVS, as it should only contain source files and binary library files.

3rd party libraries can be uploaded to CVS (together with their licences) provided that a version number is used as part of their file name or set in the Eclipse manifest if it is delivered as a “wrapped” Eclipse plug-in. Versions of popular libraries can vary enormously. JDOM 1.0 is very different to JDOM 0.9, for example. The same goes for Tomcat - some things work fine for 5.5, but not for version 5.0. Name the file jdom-1.0.jar, not jdom.jar.

Note that modules cannot be deleted permanently by users. For this to happen a formal request has to be submitted to the SourceForge technical support team by a project administrator. This process can take a few days to complete.

6.3.9. House Rules

In order to maintain good housekeeping for using the CVS repository in the project the following rules have been put in place:

1. Committed code should at all times be able to be compiled.
2. Committed code should at all times be able to be run.
3. Developers should check out the latest code from CVS before committing their own changes
4. Any code conflicts should be resolved between developers preferably informally, but failing that via a project co-ordinator
5. Folders (a.k.a "modules") should **not** be created under CVSROOT but only directly under HEAD. See above for more details as to folder structure.
6. Module naming conventions should be followed, see above
7. JUnit tests should be given their own modules, see above.
8. Compiled class files and the compiled output folder (usually “bin”) should not be added to the CVS. Only source files and libraries can be uploaded.
9. Library files should be named according to their version and/or the Eclipse manifest version number set accordingly.

6.3.10. Component Owners

Each component or other coherent piece of software in the TENCompetence system will be assigned a Component Owner. The Component Owner is the organisation, university, or company that created it. When a component requires changes from a non-Component Owner, any changes should be discussed with the Component Owner first. The Component Owner decides how to handle changes. For example, a developer working on a QTI tool that requires changes to the central data API, would have to contact Harrie Martens or Hubert Vogten at OUNL. Depending on the required changes Harrie or Hubert could either

- update the central data API, or
- allow the developer to make the changes herself, or
- allow the developer to submit a patch file

6.4. Coding conventions and guidelines

Note - the following are modelled on the Eclipse / Sun coding conventions and guidelines.

6.4.1. Naming conventions

Classes and Interfaces

Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep the class names simple and descriptive. Use whole words - avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

Examples:

- class Raster;
- class ImageSprite;

Interface names should be capitalized like class names. For interface names, we follow the "I"-for-interface convention: all interface names are prefixed with an "I". For example, "IWorkspace" or "IIndex". This convention aids code readability by making interface names more readily recognizable.

Additional rules:

The names of exception classes (subclasses of Exception) should follow the common practice of ending in "Exception".

Methods

Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Examples:

- run();
- runFast();
- getBackground();

Additional rules:

The naming of methods should follow common practice for naming getters (getX()), setters (setX()), and predicates (isX(), hasX()).

Variables
Instance, static, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with dollar sign $ characters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic - that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

Examples:

- int i;
- char c;
- float myWidth;

Constants
The names of class constants and of ANSI constants should be all uppercase with words separated by underscores ("_").

Examples:

- static final int MIN_WIDTH = 4;
- static final int MAX_WIDTH = 999;
- static final int GET_THE_CPU = 1;

Fields
Class global Field variables should start with the prefix “f”.

Examples:

- private String fName;
- protected int fCount = 1;

Plug-ins and Extension Points
All plug-ins (and plug-in fragments) must have unique identifiers following the same style of naming convention as Java packages. For example, the Eclipse workbench plug-in is named org.eclipse.ui.

Extension points that expect multiple extensions should have plural names. For example, "builders" rather than "builder".

Further guidelines as specified by Sun can be found at [22].

6.4.2. Conventions for plug-ins

Shared code should be packaged into plug-ins so that each plug-in offers a unique set of functionality allowing for optimal re-use.

Dependencies between UI components and business logic components should be eliminated. For example, utility type plug-ins should be factored such that they do not depend on or reference UI components. A Model-View-Controller architecture and Event Model facilitates this pattern. References to Java Interfaces rather than classes also contributes to the plug-in architecture.

Care should be taken when exporting packages and classes. Public packages and classes can be declared in the plug-in manifest and these should usually be limited to public API packages. All other implementations can be made private. This requires the separation of the API from internal implementation. By exporting only API classes one can limit the visibility to downstream clients. Negotiation between plug-in developers may be necessary in order to determine the right balance between public and private dependencies, and developers should reveal only the API in which they have confidence, but be prepared to reveal more of the API as clients ask for it.

Re-usable third-party libraries may be wrapped in plug-in manifests for deployment as Eclipse plug-ins and named accordingly. For example, the XML library JDOM 1.0 can be packaged with a manifest file and made available as *org.jdom.plugin* version 1.0.0. Plug-ins that depend on a particular third-party library may instead ship the required library file(s) in the packaged plug-in "as is" (usually in a "lib" folder).

JUnit tests for each component should be provided in their own plug-in modules – see section 6.2 for details.

Extension Points

Developers are encouraged to provide and declare Extension Points in their plug-in manifests. This Eclipse mechanism provides an XML-based schema approach for declaring the mechanics of a plug-in's extension point(s).

Extension points define new function points for the platform or application that other plug-ins can plug into. An Extension Point is declared in XML-based schema file that defines the grammar that formally expresses elements, attributes, and types that it relates to. This information can be used by plug-ins to validate extensions. An example of an Eclipse Extension Point is a Menu Item, a View, or an Editor.

Future plug-in contributors may then leverage these extension points to provide further functionality to the client.

Naming conventions

TENCompetence plug-ins must follow the naming convention of project namespace followed by module name. The project namespace is `org.tencompetence.` TENCompetence client plug-ins have the "client" name appended to the namespace. For example, a new plug-in for the client application would be named `org.tencompetence.client.myplugin.`

The convention for naming private and public API packages is for every plug-in intended for external use to have two name spaces. If a package name contains `internal`, the classes it contains are not intended to be used publicly.

Eclipse plug-in version numbers should follow the convention x.x.x.qualifier (for example, “1.0.1.qualifier”). The “qualifier” part will ensure that the resulting plug-in filename is suffixed with the current time-stamp.

6.4.3. User Interface Guidelines

These guidelines are modelled after the Eclipse User Interface Guidelines and, being too lengthy to reprint, can be found at:

http://wiki.eclipse.org/index.php/User_Interface_Guidelines

6.4.4. Runtime, Platforms, Tools and Libraries

Java 5 (v1.5.x) is to be used for the SDK and target runtime. Java 6 (v1.6.x) may be used at a later date once it has been firmly proved and established on all platforms (Windows, Linux, Mac). As of the time of writing, Java 6 is not supported on Apple OS X.

As of the time of writing Eclipse version 3.3.1 is employed both as SDK and target platform for the PCM client and other Eclipse-based applications. It is recommended to use the publically available release version (“Europa”) and its associated plug-ins.

The following table lists the **Eclipse tools and libraries** that are currently used for the Eclipse client PCM and its plug-ins:

Name	Version	Description
Eclipse Platform	Europa	Eclipse SDK and RCP
Eclipse Web Tools	Europa	Eclipse Web Tools plug-ins
GEF	Europa	Graphical Editing Framework
ECF	Europa	Eclipse Communication

Table 1. Eclipse tools and libraries used in the project

7. PCM services reference

7.1. Entity services

7.1.1. General rules

General - formatting

- General: dates are formatted according to the ISO 8601 standard

General – rules

- Use of Create / Read / Update / Delete services is only allowed for objects within a Community to which the user is registered.
- It's never allowed to move an object from one Community to another.
- The XML for an update message (PUT) only contains the mandatory fields and the fields that need updating.
- Sharing: it's only allowed to relax the sharing, never to restrict it further.
  The possible settings for a community are, from strict to relaxed: private => closed => open.
  The possible settings for other objects are, from strict to relaxed: 'do not share' => 'share; only I and the following people can change' (in this case, the creator can assign update rights to a specific group of persons) => 'share; everyone in the community can change'.
- It's only allowed to delete an object if it's not in use by any other remaining object (registered user, a linked CDP, …).
- An object can only be deleted by its creator.

Common fields

- <title> contains the title as displayed on the screen for an object. As the title is shown on a number of places, it is advised to keep the title short and concise.
- <description> contains a detailed description of what an object is about. It's possible to use a long description.
- <creator> contains the identification of the user that created an object.

7.1.2. Structure for service descriptions

The descriptions for the services is grouped per entity (Community, Competence, …). For each entity, the following parts are explained:

- Description: explains in a few lines what the entity is used for within the PCM. Check the domain model for a precise and detailed definition.
- Services: the (low level) services available for the entity.
- URI format: the structure to which the URI needs to conform for this entity.
- XML – structure: graphical view of the structure (.xsd) for the XML messages of this entity and a table providing additional information about individual fields.
  To reduce the number of pictures, the root of each message is left out. The "XML – allowed elements and …" section can be used to determine the root element's tag name.
- XML – allowed elements and example: an example of an XML message and shows in which requests the different elements can be used. The letters "C" (Create) "R" (Read) and "U" (Update) from "CRUD" are used to indicate this.

Note: “D” is not used, as deleting an object doesn’t require an XML message body. Sending an HTTP DELETE to a URI is enough.

7.1.3. Assessment Activity

Description

Classical multiple choice tests to test whether you master certain principles.

In this case the assessment activity can contain one or more assessment items, including the mechanism to score the test (only straightforward scoring by measuring the percentage correct answers).

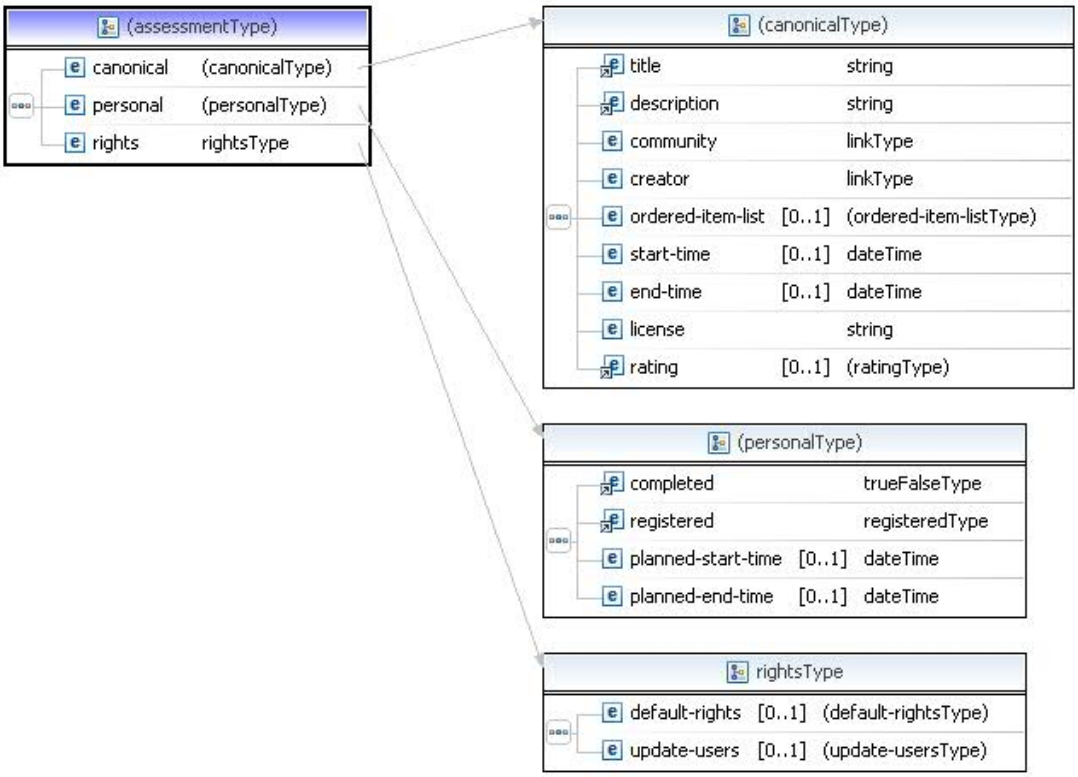
Services

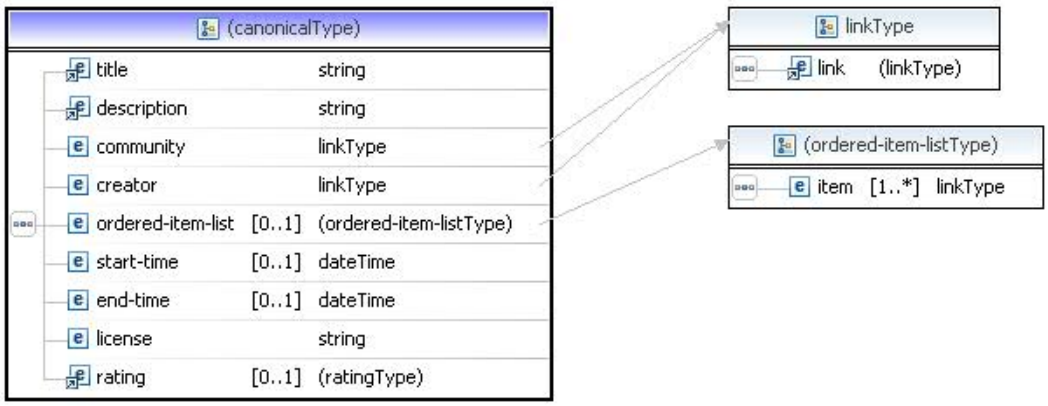
- Create
- Retrieve
- Update
- Delete
- Share
  Sending a PUT message with a /rights element will modify the share settings for the Assessment activity to the rights as passed in the message.
- Register
  Sending a PUT message with a new value for the /personal/registered element will register or unregister the current user for an Assessment activity.
- Complete
  Sending a PUT message with a new value for the /personal/completed element will mark an Assessment activity as complete or incomplete for the current user.

URI format

http://{server}/{community-guid}.community/{assessment-guid}.assessment
http://{server}/{community-guid}.community/{assessment-guid}.assessment?statistics=true

XML - structure





Path	Description
/canonical/community	The community to which the Assessment belongs. When creating an Assessment, its community is deduced from the URI.
/canonical/ordered-item-list	Contains the identification of the Assessment Items (questions) that form this Assessment.
/canonical/start-time	The date and time on which the Assessment is planned to be available to Learners.
/canonical/end-time	The date and time on which the Assessment is planned to be made unavailable to Learners.
/canonical/license	The license text (e.g. a Creative Commons text) applying to the Assessment.
/personal/planned-start-time	The date and time on which the user intends to start the Assessment work.
/personal/planned-end-time	The date and time on which the user intends to finish the Assessment work.

XML - allowed elements and example

```
CRU <assessment xml:space="preserve" xmlns="http://www.tencompetence.org/api/v1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.tencompetence.org/api/v1.0
http://www.tencompetence.org/api/v1.0">
CRU    <canonical>
CRU          <title>xxxxx</title>
CRU          <description>xxxx</description>
CR           <type>type of activity</type>
 R           <community>
                  <link type=”community” href=”xxxx”>Display title of the
community</link>
            </community>
 R          <creator>
                  <link type=”user” href=”xxxxxxx”>display title</link>
            </creator>
CRU          <ordered-item-list>
                  <item>
                        <link type=”item href=”xxxxxxx”>display title</link>
                  </item>
                  <item>
                        <link type=”item href=”xxxxxxx”>display title</link>
                  </item>
            </ordered-item-list >
CRU          <start-time>xxxxx</start-time>
CRU          <end-time>xxxxx</end-time>
CRU          <license>zzzzzz</license>
            {
 R                <rating>0.nnn</rating>
            } *
      </canonical>

CRU   <personal>
CRU          <completed>true/false</completed>
CRU          <registered>notregistered/registered/pending/rejected</registered>
CRU          <planned-start-time>xxxxxx</planned-start-time>
CRU          <planned-end-time>xxxxxx</planned-end-time>
      </personal>

CRU   <rights>
            ..
      </rights>
</assessment>
```

* Returned when doing a GET with parameter statistics=true.

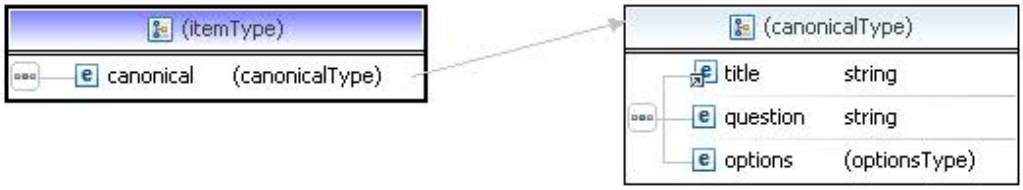
7.1.4. Assessment Item

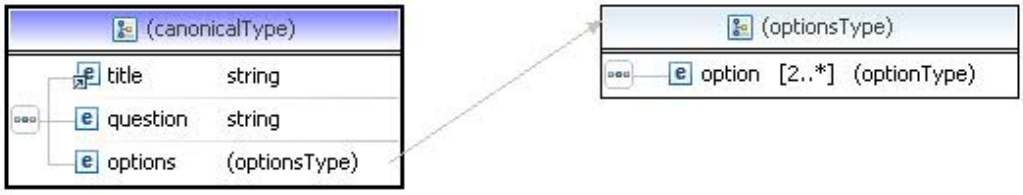
Description
One multiple choice question, that can be used in a number of Assessment Activities.

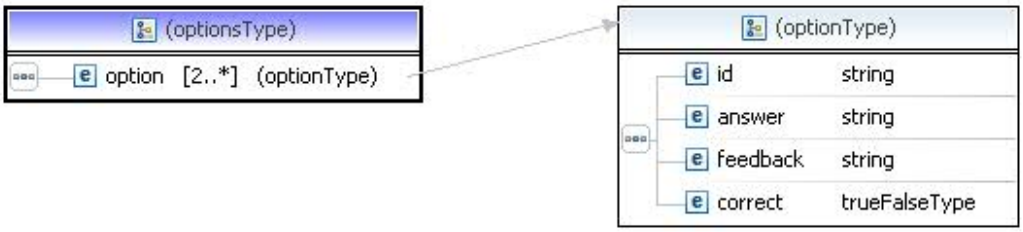
Services
- Create
- Retrieve
- Update
- Delete
- Retrieve list: summary info for all assessment items from a specific community. See List Assessment Items.

URI format
http://{server}/{community-guid}.community/{item-guid}.item

XML - structure







Path	Description
/canonical/question	The text for the question.
/canonical/options	The different answering options.
/canonical/options/option/answer	The text for one of the options.
/canonical/options/option/feedback	Information to explain what is right or wrong about this option.
/canonical/options/option/correct	Indicates if this is the correct answer to the question. Exactly one of the options has to be the correct answer to the question.

XML - allowed elements and example

```
CRU    <item xml:space="preserve" xmlns="http://www.tencompetence.org/api/v1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.tencompetence.org/api/v1.0
http://www.tencompetence.org/api/v1.0">
CRU    <canonical>
CRU           <title>display title</title>
CRU           <question>How many cm go into one m?</question>
CRU           <options>
CRU                  <option>
CRU                        <id>xxxx</id>
CRU                        <answer>100</answer>
CRU                        <feedback>Well done!</feedback>
CRU                        <correct>true</correct>
                     </option>
                     <option>
                           <id>xxxx </id>
                           <answer>90.48</answer>
                           <feedback>Sorry, .. </feedback>
                           <correct>false</correct>
                     </option>
              </options>
       </canonical>
</item>
```

7.1.5. Community

Description

A domain representing a certain profession. Users can collaborate within the context of a certain community.

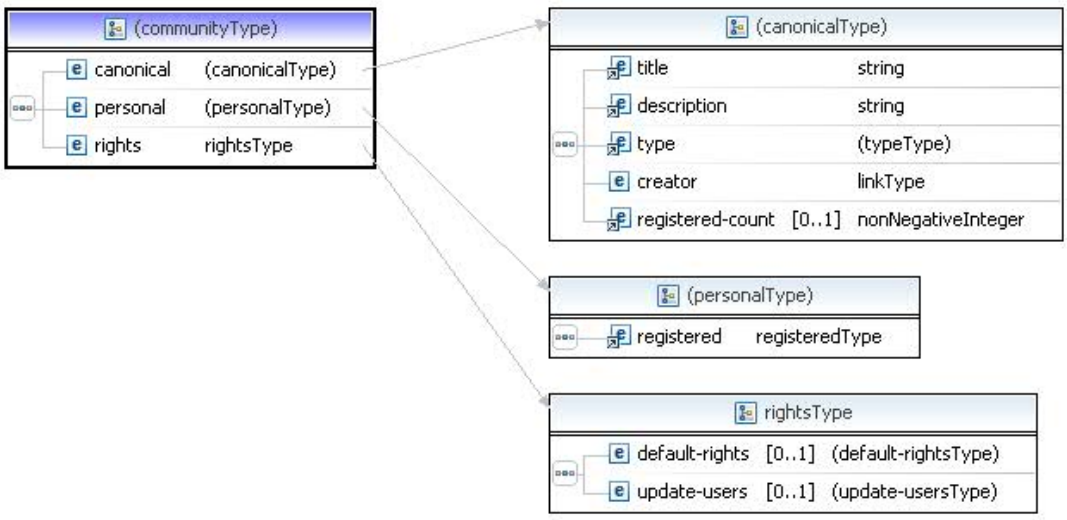
Services

- Create
- Update
- Delete
- Retrieve
- Retrieve list: summary info for all communities
- Share: private (accessible only by creator), open (accessible by everyone), closed (accessible after permission)
- Request for Community registration
- Approve of Community registration request
- Rejection of Community registration request

URI format

http://{server}/{community-guid}.community
http://{server}/{community-guid}.community&statistics=true

XML - structure



Path	Description
/canonical/type	Registration type of the community:'open' (everybody can join without approval), 'closed' (people can only join after approval of an administrator) or 'private' (nobody can join, the community is only accessible for its creator).
/canonical/registered-count	The number of persons that are registered for the community.

XML - allowed elements and example

```
CRU    <community>
CRU    <canonical>
CRU           <title>Java Programmers</title>
CRU           <description>A Community for Java programmers…</description>
CRU           <type>open/closed/private</type>
 R            <creator>
 R                  <link type="user" href="xxxxxxx">display title</link>
              </creator>
              {
 R            <registered-count>nnnn</registered-count>
              }*
       </canonical>

CRU    <personal>
CRU           <registered>notregistered/registered/pending/rejected</registered>
       </personal>

CRU    <rights>
              ..
       </rights>
</community>
```

* Returned when doing a GET with the option statistics=true.

7.1.6. Competence

Description

A competence is defined as the ability (´disposition´) of an actor to act effectively and efficiently upon the events in an ecological niche (an occupation, a hobby, a market, a sport, etc.). In short: the ability to perform effectively in a situation.

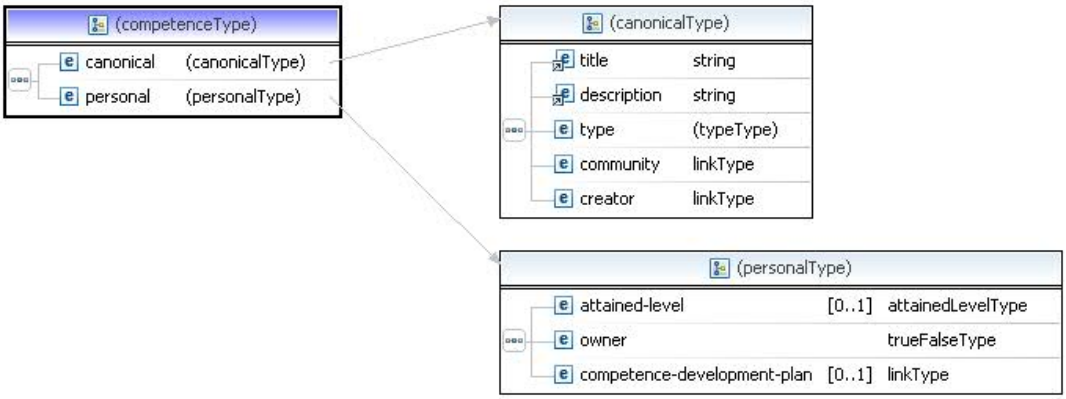
Services

- Create
  Rules:
  - A Competence needs to be created within an existing Community.
- Retrieve
- Update
- Delete
- Retrieve list: summary info for all competences from a specific community

URI format

http://{server}/{community-guid}.community/{competence-guid}.competence

XML - structure



Path	Description
/canonical/type	Allowed values are the 5 Cheetam & Chivers types: 'personal', 'functional', 'cognitive', 'ethical' and 'transmeta'.
/personal/attained-level	The current competence level of the user for this Competence. This can also be used to update the user's competence level.
/personal/owner	Specifies if the user is the owner of this Competence, this is mainly used when retrieving the list of Competences.
/personal/competence-development-plan	Identification of the CDP that the user has selected to work on this Competence.

XML - allowed elements and example

```
CRU <competence>
CRU    <canonical>
CRU          <title>Java Programming</title>
CRU          <description>Describes the ability to create…</description>
CRU          <type>ethical</type>
 R           <community>
 R                 <link type="community" href="xxxx">Display title of the
community</link>
             </community>
 R           <creator>
 R                 <link type="user" href="xxxxxxx">display title of the
competence</link>
             </creator>
       </canonical>

CRU    <personal>
CRU          <attained-level>0..8</attained-level>
CRU          <owner>true/false</owner>
 R           <competence-development-plan>
 R                 <link type ="competence-development-plan" href="url">display
title</link>
             </competence-development-plan>
       </personal>
</competence>
```

Competence Development Plan (CDP)

Description

A Competence Development Programme (CDP; synonyms: route, learning path, curriculum, programme) is an ordered set of activities and units of learning that have to be (or are) followed to attain a certain level for a competence.

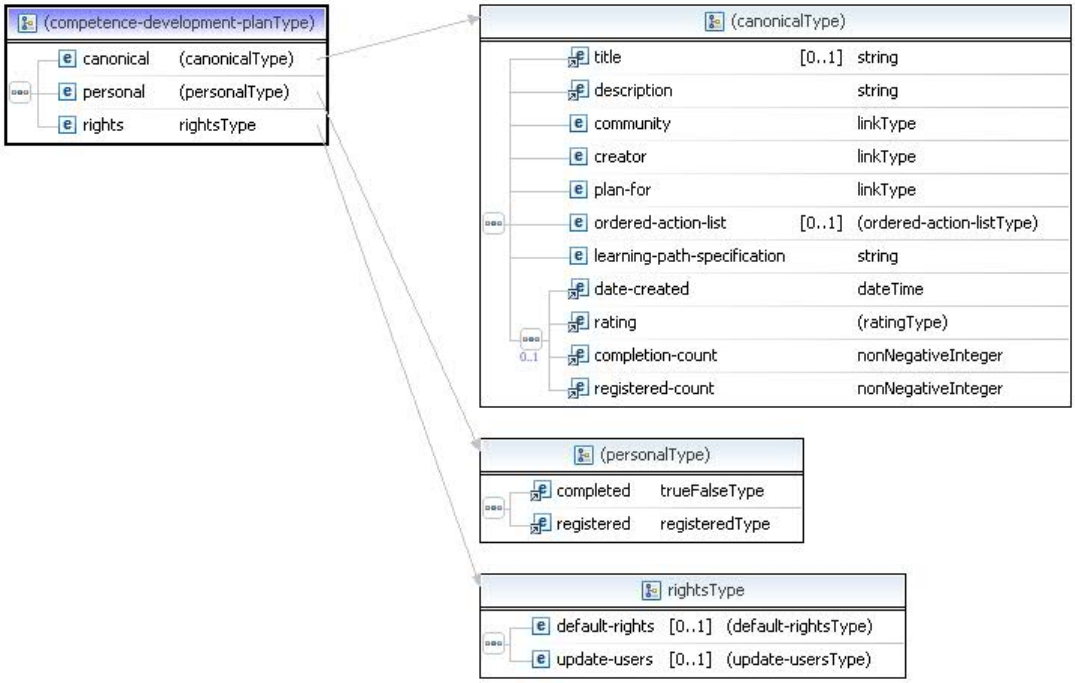
Services

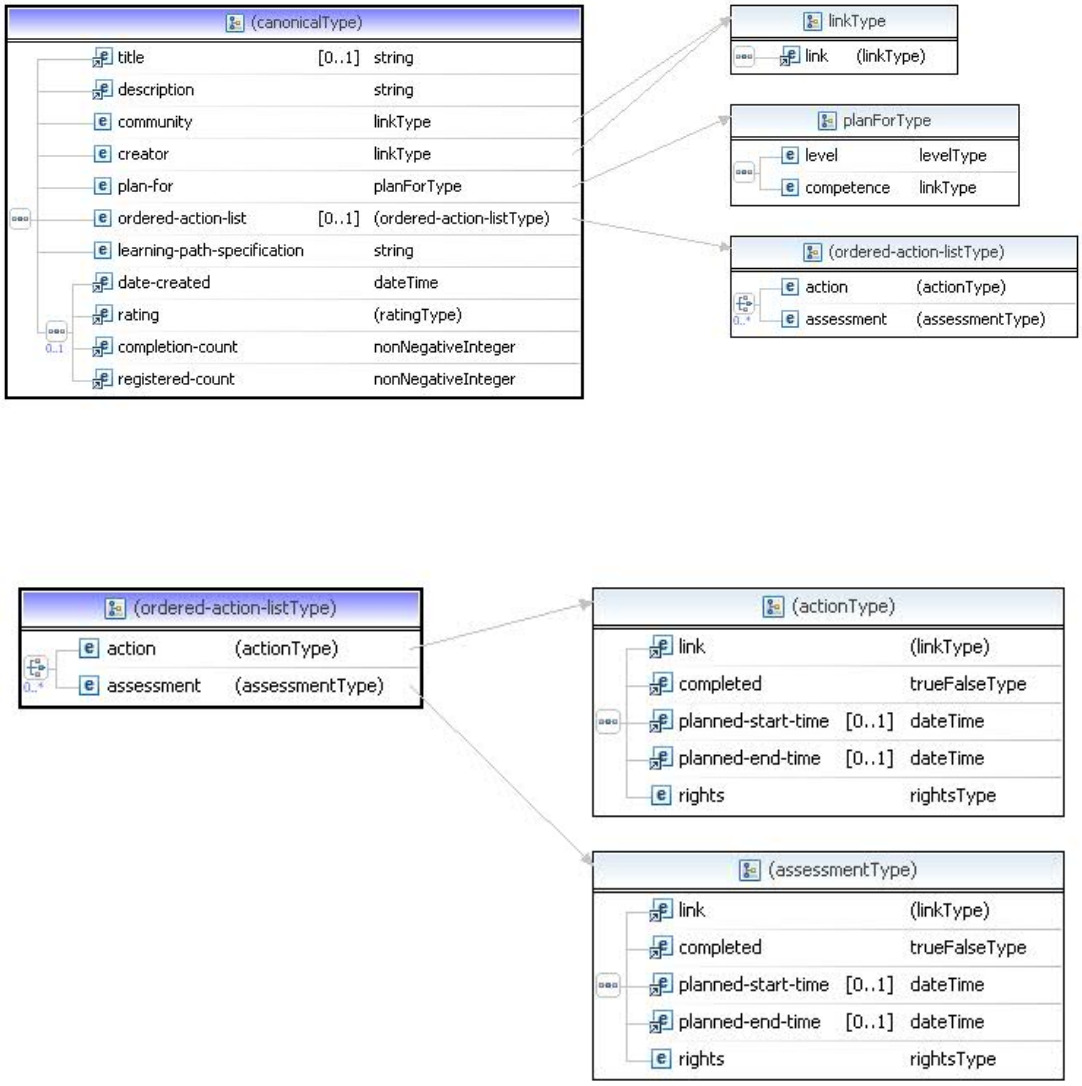
- Create
- Update
- Delete
- Retrieve
- Retrieve list: summary info for all CDPs from a specific community
- Share
- Register
- Complete

URI format

http://{server}/{community-guid}.community/{competence-development-plan-guid}.cdp
http://{server}/{community-guid}.community/{competence-development-plan-guid}.cdp&statistics=true

XML - structure





Path	Description
/canonical/plan-for/level	The competence level for this CDP. When successfully completing the CDP, the user's attained level on the linked Competence is increased to the CDP's level.
/canonical/plan-for/competence	The Competence to which this plan applies.
/canonical/ordered-action-list	The actions to perform for the CDP. The ordering of actions in this list is persisted.
/canonical/ordered-action-list/action/planned-start-time	The date and time on which the current user intends to start their work on this Action.
/canonical/ordered-action-list/action/planned-end-time	The date and time on which the current user intends to finish the work on this Action.
/canonical/ordered-action-list/assessment/planned-start-time	The date and time on which the current user intends to start their work on this Assessment.
/canonical/ordered-action-list/assessment/planned-end-time	The date and time on which the current user intends to finish the work on this Assessment.

Path	Description
/canonical/ordered-action-list/action/completed	Indicates if the current user has completed the action. This is a deviation of the standard, which uses the /personal tag for information about the current user.
/canonical/learning-path-specification	An XML string containing the specification for the advised learning path through the CDP's action. Section 7.1.8 Learning path describes the format etc. of the learning path.
/canonical/date-created	The date and time on which the CDP was originally created.
/canonical/rating	Float with range [0..1] indicating the average rating given to this CDP.
/canonical/completion-count	The number of persons that have completed this CDP.
/canonical/registered-count	The number of persons that are registered for this CDP.
/personal/completed	Indicates if the user has completed this CDP. Competence Profiles to which the user is registered, that have the CDP's Competence within them, are automatically updated to the user. When the Competence in those Competence Profiles has a level lower or equal than the CDP's level, they are marked as attained. E.g. a user completes a CDP of level 5 for Competence 'Communication skills'. His Competence Profiles 'HR Manager' and 'Project Manager', both containing 'Communication skills' at level 5 will show 'Communication Skills' as attained. A CDP can only be completed by setting the /personal/completed field to 'true'. It's not automatically completed by completing all contained actions.

XML - allowed elements and example

```
CRU <competence-development-plan>
CRU    <canonical>
CRU           <title>xxxxx</title>
CRU           <description>xxxx</description>
 R            <community>
 R                   <link type=”community” href=”xxxx”>Display title of the
community</link>
              </community>
 R            <creator>
 R                   <link type=”user” href=”xxxxxxx”>display title</link>
              </creator>
CRU           <plan-for>
CRU                 <level>3</level>
CR                  <competence>
CR                        <link type=”competence” href=”url”>display title</link>
                    </competence>
              </plan-for>

CRU           <ordered-action-list>
CRU                 <action>
CRU                       <link type=”action” href=”url”>display title</link>
 R                        <completed>true/false</completed>
CRU                       <planned-start-time>xxxxxx</planned-start-time>
CRU                       <planned-end-time>xxxxxx</planned-end-time>
 R                        <rights>
                                ..
                          </rights>

                    </action>
                    <action>
                          <link type=”action” href=”url”>display title</link>
                          <completed>true/false</completed>
                          <planned-start-time>xxxxxx</planned-start-time>
                          <planned-end-time>xxxxxx</planned-end-time>
                          <rights>
                                ..
                          </rights>
                    </action>
              </ordered-action-list>
CRU           <learning-path-specification>
                    ..
              </learning-path-specification>
              {
 R            <date-created>a date</date-created>
 R            <rating>0.nnn</rating>
 R            <completion-count>nnn</completion-count>
 R            <registered-count>nnn</registered-count>
              } *
       </canonical>

CRU    <personal>
CRU           <registered>notregistered/registered/pending/rejected</registered>
       </personal>

CRU    <rights>
              ..
       </rights>
</competence-development-plan>
```

* This is only returned doing a GET with the option statistics=true

7.1.7. Learning path

Description

The learning path describes the order of the actions within a CDP and a visual representation of these actions, to enable a client to display the path. A learning path is stored as a part of a CDP, it is not a separate service.

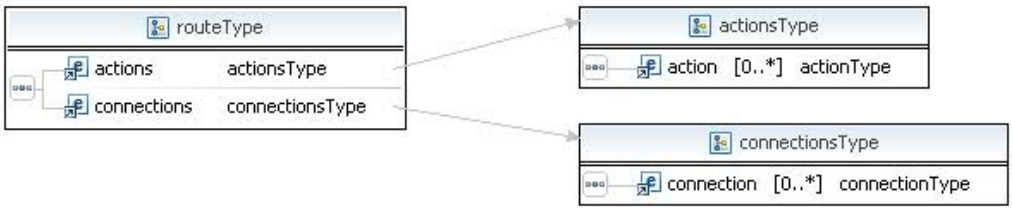
Services

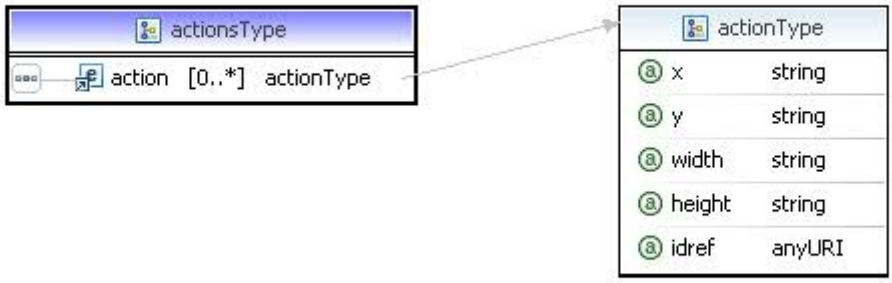
Not applicable, because it is an integrated part of the CDP services.

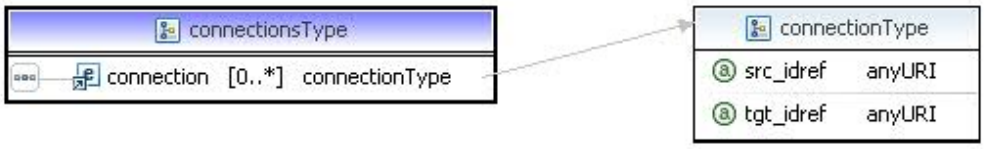
URI Format

Not applicable.

XML structure







Path	Description
/actions/action.x	The object's x-coordinate in the visual representation of the learning path.
/actions/action.y	The object's y-coordinate in the visual representation of the learning path.
/actions/action.width	The object's width in the visual representation of the learning path.
/actions/action.height	The object's height in the visual representation of the learning path.
/actions/action.idref	Identification of the object.
/connections/connection.src_idref	Identification of the source object for a connection between two objects.
/connections/connection.tgt_idref	Identification of the target object for a connection between two objects. Identification of the target object.

XML – example

```
<route>
  <actions>
       <action x="109" y="40" width="40" height="20"
              idref="http://../TENCServer/cu1.community/a668.action" />
       <action x="109" y="156" width="40" height="20"
              idref="http://../TENCServer/cu1.community/634bf6cb.action" />
  </actions>
  <connections>
       <connection src_idref="http://../TENCServer/cu1.community/902cde2e.action"
               tgt_idref="http://../TENCServer/cu1.community/bf6cbcc9f.action" />
  </connections>
</route>
```

Notes:

- Updating a learning path is done by replacing the full XML string, a partial update is not possible.
- The fields used during a create and retrieve are exactly the same.

7.1.8. Competence Profile

Description

A set of competences that define the minimum requirements for a specific function/job. The competences that are specified under each function specify the target competence levels of that competence that must be met in order to obtain the function.

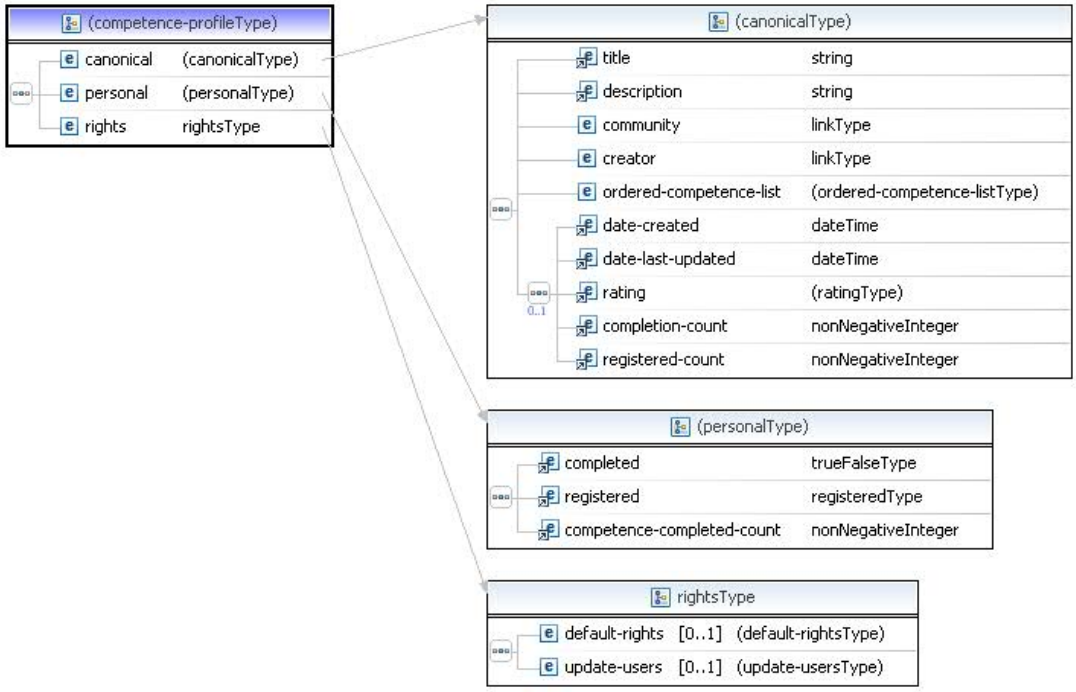
Services

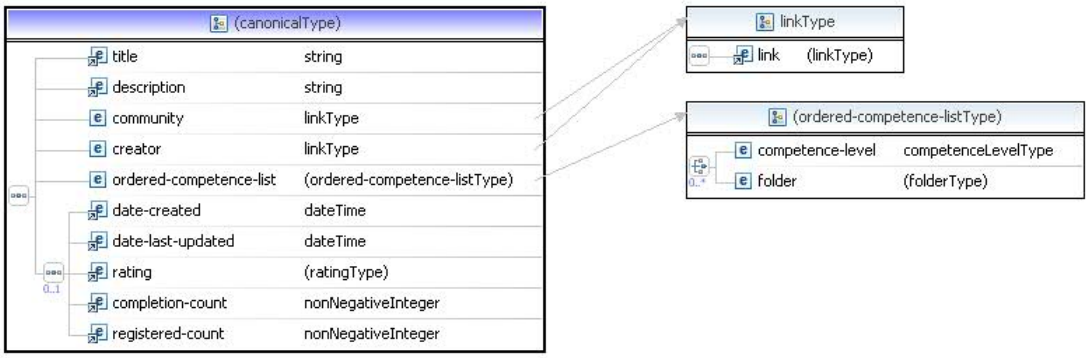
- Create
- Update
- Delete
- Retrieve
- Retrieve list: summary info for all profiles from a specific community
- Share: not shared (accessible only by creator), shared - updates by group (accessible by the whole community, updates only by a named group and the creator), sharedforupdate – edit by community (accessible by the whole community, updates by anyone from the community)
- Register
  When registered to a certain Community, the user is allowed to register to all shared Competence Profiles of that Community.
- Complete
  A user can set each Competence Profile, to which they are registered, to completed.

URI Format

http://{server}/{community-guid}.community/{competenceprofile-guid}.competenceprofile

XML structure





Path	Description
/canonical/community	Identification of the Community in which the Competence Profile is contained.
/canonical/ordered-competence-list	An ordered list of the competences, including their competence level, that form this Competence Profile. The list is persisted in the specified order.
/canonical/rating	Float with range [0..1] indicating the average rating given to this Competence Profile.
/canonical/completion-count	The number of persons that have completed the Competence Profile.

Path	Description
/canonical/registered-count	The number of persons that have registered for this Competence Profile.
/personal/completed	Indicates if the user has completed this Competence Profile: ‘true’ or ‘false’.
/personal/competence-completed-count	The number of competences referenced in this profile that the user has completed.

XML – allowed elements and example

```
<competence-profile>
CRU    <canonical>
CRU           <title>xxxxxxxx</title>
CRU           <description>xxxx</description>
CR            <community>
CR                  <link type=”community” href=”http://xxxx”>Display title of the
community</link>
              </community>
 R            <creator>**
                    <link type=”user” href=”http://xxxxxxx”>display title</link>
              </creator>
CRU           <ordered-competence-list>
CRU                 <competence-level>
CRU                       <level>3</level>
                          {
 R                        <completion-count>nnn</completion-count>
              }*
CRU                       <competence>
CRU                             <link type=”competence”
href=”http://...url”>display title</link>
 R                              <attained-level>2</attained-level>
                          </competence>
                    </competence-level>
CRU                 <folder>
CRU                       <title>xxxxxx</title>
CRU                             <competence-level>
CRU                             <level>3</level>
                                {
 R                              <completion-count>nnn</completion-count>
                                }*
CRU                             <competence>
CRU                                   <link type=”competence”
href=”http://..url”>display title</link>
 R                                    <attained-level>1</attained-level>
                                </competence>
                          </competence-level>
                    </folder>
              </ ordered-competence-list >
              {
 R            <date-created>a date</date-created>
 R            <date-last-updated>a date</date-last-updated>
 R            <rating>a float between 0.0 and 1.0</rating>
 R            <completion-count>nnnn</completion-count>
 R            <registered-count>nnnn</registered-count>
              }*
       </canonical>

CRU    <personal>
              <completed>true/false</completed>
              <registered>notregistered/registered/pending/rejected</registered>
       </personal>

       <rights> **
              ...
       </rights>
</competence-profile>
```

* Only returned when parameter “statistics=true” is provided.
** See the section on rights for details.

7.1.9. Learning Activity

Description

Learning activities are tasks for learners that describe what they are advised to do in order to attain certain learning or assessment objectives, given some prerequisites.

Services

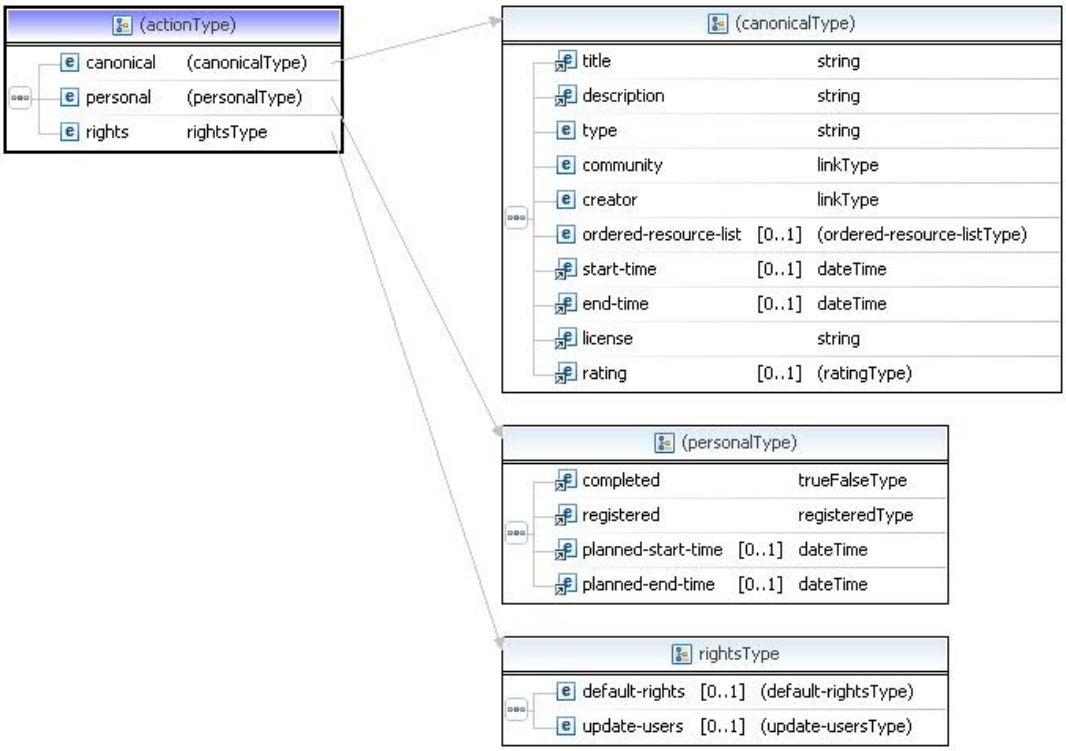
- Create
- Update
- Delete
- Retrieve
- Share
- Register
- Complete

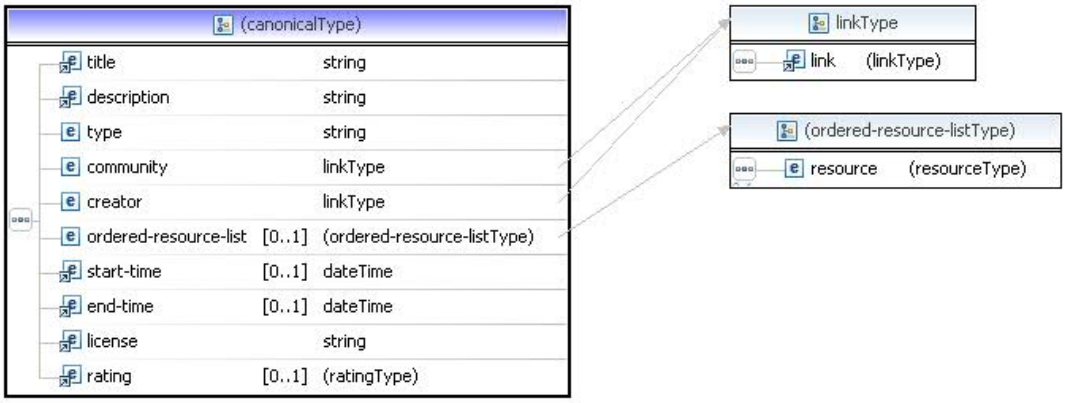
The create and update services include the possibility to create / update the route (which is shown graphically by the PCM client).

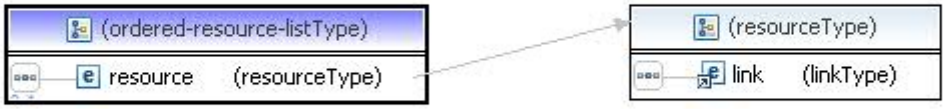
URI format

http://{server}/{community-guid}.community/{action-guid}.action
http://{server}/{community-guid}.community/{action-guid}.action&statistics=true

XML - structure







Path	Description
/canonical/type	Allowed values are the 5 Cheetam & Chivers types: 'personal', 'functional', 'cognitive', 'ethical' and 'transmeta'.
/canonical/ordered-resource-list	The resources to read / use / reference for the Action. The ordering of resources in this list is persisted.
/canonical/start-time	The date and time on which the Action is planned to be available to Learners.
/canonical/end-time	The date and time on which the Action is planned to be made unavailable to Learners.
/canonical/license	The license text (e.g. a Creative Commons text) applying to the Action.
/canonical/rating	Float with range [0..1] indicating the average rating given to this Action.
/personal/planned-start-time	The date and time on which the current user intends to start their work on this Action.
/personal/planned-end-time	The date and time on which the current user intends to finish the work on this Action.
/personal/completed	Indicates if the current user has completed the action.

XML - allowed elements and example

```
CRU    <action>
CRU    <canonical>
CRU           <title>xxxxx</title>
CRU           <description>xxxx</description>
CRU           <type>type of activity</type>
 R            <community>
 R                   <link type=”community” href=”xxxx”>Display title of the
community</link>
              </community>
 R            <creator>
                    <link type=”user” href=”xxxxxxx”>display title</link>
              </creator>
CRU           <ordered-resource-list>
CRU                 <resource>
CRU                       <link type=”resource’ href=”xxxxxxx”>display title</link>
                    </resource>
CRU                 <resource>
                          <link type=”resource’ href=”xxxxxxx”>display title</link>
                    </resource>
              </ordered-resource-list >
CRU           <start-time>xxxxx</start-time>
CRU           <end-time>xxxxx</end-time>
CRU           <license>zzzzzz</license>
              {
 R            <rating>0.nnn</rating>
              } *
       </canonical>

CRU    <personal>
CRU           <registered>notregistered/registered/pending/rejected</registered>
CRU           <completed>true/false</completed>
CRU           <planned-start-time>xxxxxx</planned-start-time>
CRU           <planned-end-time>xxxxxx</planned-end-time>
       </personal>

CRU    <rights>
              ..
       </rights>
</action>
```

* This is only returned doing a GET with the option statistics=true

7.1.10. Resource

Description

Any kind of resources that can be used in learning. Typical resources are:
1. HTML pages
2. Podcasts / Vodcasts
3. Digital documents
4. Computer programs

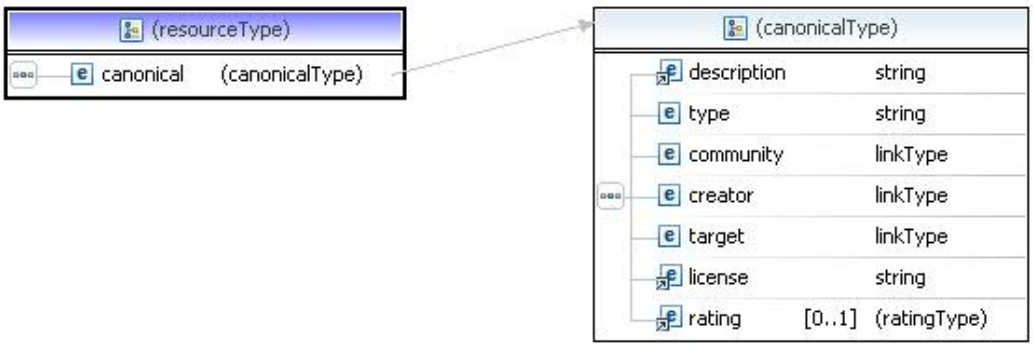
Services

- Create web link
- Create file (uploads a file to the TENC server)
- Update metadata
- Delete
- Retrieve metadata
- Retrieve file
- Retrieve list: summary info for all resources from a specific community

URI format

http://{server}/{community-guid}.community/{resource-guid}.resource
http://{server}/{community-guid}.community/{resource-guid}.resource&statistics=true

XML - structure



Path	Description
/canonical/type	'Web resource' (for URI's) or 'File resource' (for resources containing a body).
/canonical/community	The community in which this Resource resides.
/canonical/target	The URI for the Resource's body.
/canonical/license	The license text (e.g. an Creative Commons text) applying to the Resource.
/canonical/rating	Float with range [0..1] indicating the average rating given to this Resource.

XML - allowed elements and example

```
CRU    <resource>
CRU    <canonical>
CRU           <description>fdsfkjs skjfdskdjf jdsfask dfsfs</description>
CRU           <type>xxxx</type>
 R            <community>
                     <link type=”community” href=”xxxx”>Display title of the
community</link>
              </community>
 R            <creator>
 R                   <link type=”user” href=”xxxxxxx”>display title</link>
              </creator>
CRU           <target>
CRU                 <link type=”web-resource’ href=”xxxxxxx”>display title</link>
              <target>
CRU           <license>dfsdff</license>
              {
 R            <rating>0.nnn</rating>
              } *
       </canonical>
</resource>
```

* This is only returned doing a GET with the option statistics=true

7.1.11. User

Description

A person who uses the PCM system.

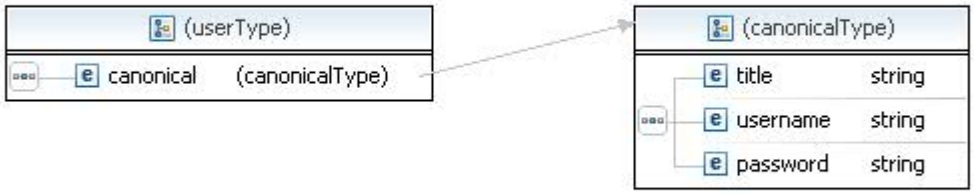
Services

- Create
- Update
  Rules:
  - Users are only allowed to update their own user account.
  - It's not allowed to update the user name (/canonical/username).
- Delete
  Rules:
  - It's only allowed to delete a user when that user is not the creator of any remaining object (CDP, Competence Profile, Community, …).
- Retrieve
- Retrieve list: summary info for all users from a specific community
  Addition of parameter "availability=true" will add an on-line indicator for each user to the result.

URI format

http://{server:port/path}/user/{user-guid}.user

XML - structure



Path	Description
/canonical/title	Display name shown to other users.
/canonical/username	Account name of the user.
/canonical/password	PCM password for the user.

XML – allowed elements and example

```
CRU <user xmlns="http://www.tencompetence.org/api/v1.0">
CRU    <canonical>
CRU           <title>Jan Janssen</title>
CR            <username>jjanssen</username>
C U           <password>france</password>
       </canonical>
</user>
```

7.2. Lists

7.2.1. List Actions

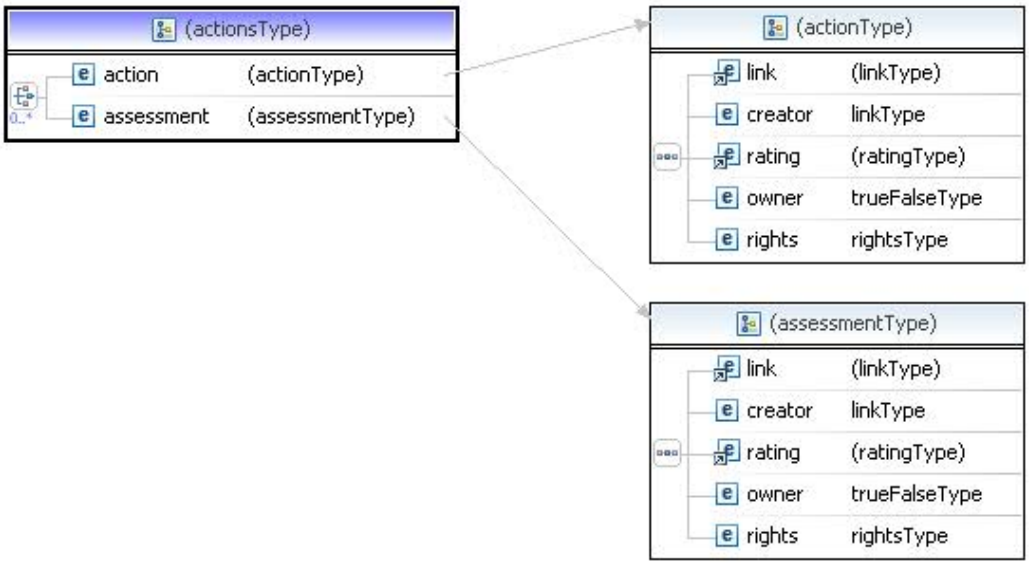
Description

Retrieves summary info for all owned and shared actions (Learning Activities and Assessment Activities) in a specific community.

URI format

http://{server}/{community-guid}.community/actions

XML - structure



Path	Description
/action/link	Identification of the Action to which this entry refers.
/action/creator	Identification of the User who created the Action.
/action/rating	Float with range [0..1] indicating the average rating given to this Action.
/action/owner	Indicates if the user is the creator of the Action
/assessment/link	Identification of the Assessment to which this entry refers.
/assessment/creator	Identification of the User who created the Assessment.
/assessment/rating	Float with range [0..1] indicating the average rating given to this Assessment.
/assessment/owner	Indicates if the user is the creator of the Assessment

XML - example

The message below is an example of the XML structure as returned by the server.

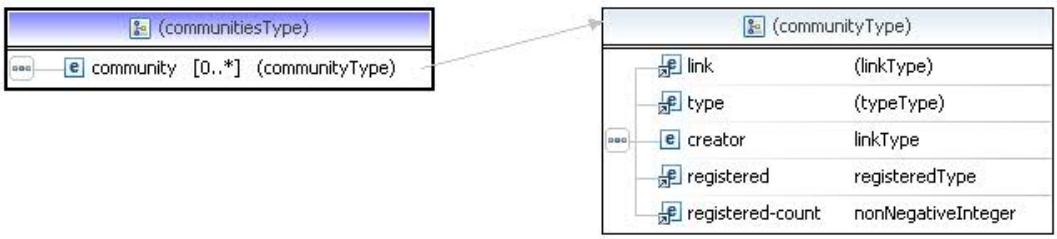
```
<actions>
	<action>
		<link type=”action” href=”xxxxxxx”>display title</link>
		<creator>
			<link type=”user” href=”xxxxxxx”>display title</link>
		</creator>
		<rating>0.nnn</rating>
		<owner>true/false</owner>
		<rights>
			..
		</rights>
	</action>

	<assessment>
		<link type=” assessment” href=”xxxxxxx”>display title</link>
		<creator>
			<link type=”user” href=”xxxxxxx”>display title</link>
		</creator>
		<rating>0.nnn</rating>
		<owner>true/false</owner>
		<rights>
			..
		</rights>
	</assessment>
</actions>
```

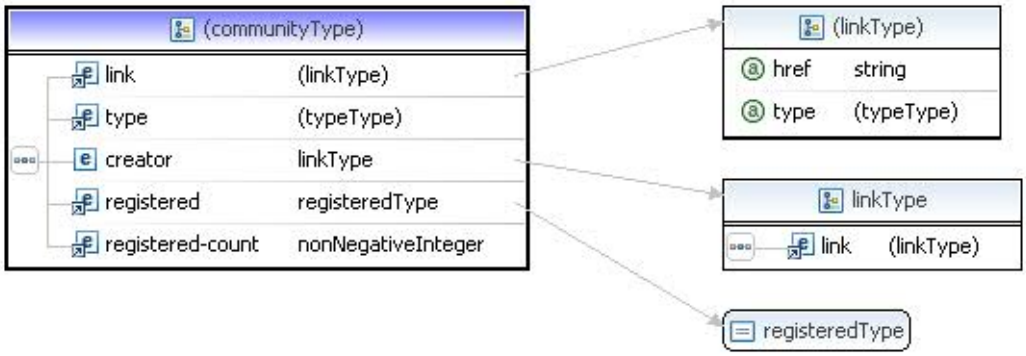
7.2.2. List Communities - authenticated

URI format

http://{server}/communities

XML - structure





Path	Description
/community/link	Identification of the Community.
/community/type	Registration type of the Community:‘open’ (everybody can join without approval), ‘closed’ (people can only join after approval of an administrator) or ‘private’ (nobody can join, the community is only accessible for its creator).
/community/registered-count	The number of persons that are registered for the Community.

XML - example

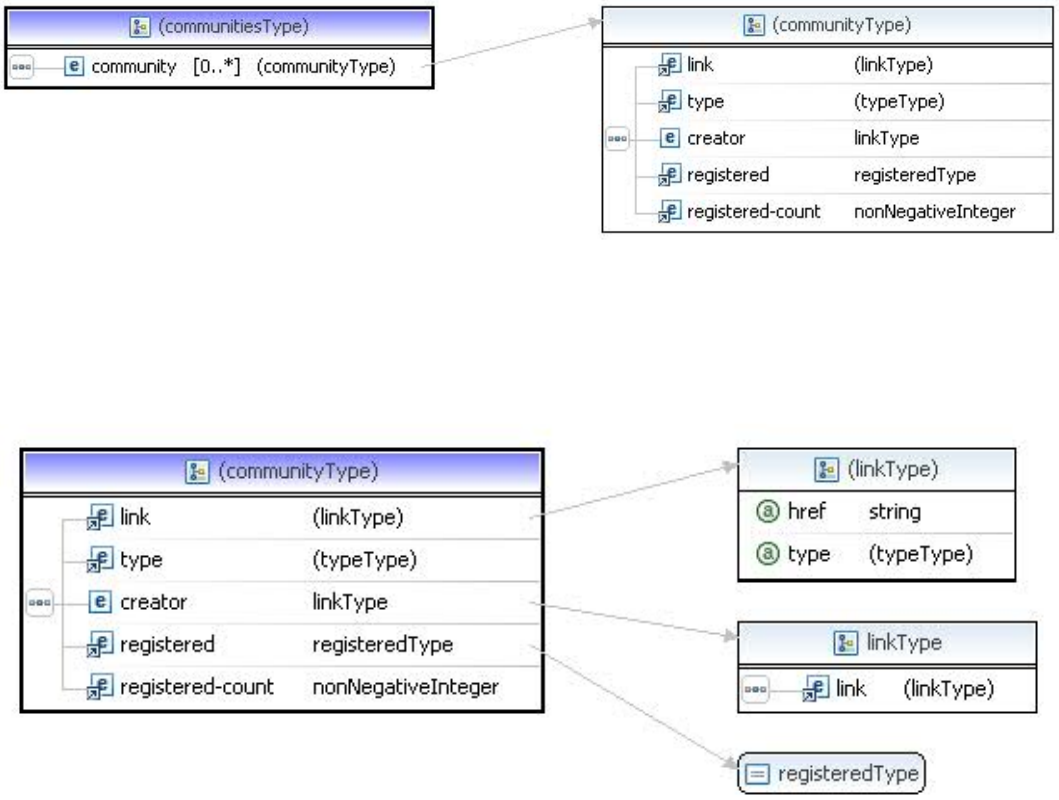
```
<communities>
	<community>
		<link type=”community” href=”xxxxxxx”>display title</link>
		<type> private/open/closed</type>
		<creator>
			<link type=”user” href=”xxxxxxx”>display title</link>
		</creator>
		<registered>notregistered/registered/pending/rejected</registered>
		<registered-count>nnnn</registered-count>
	</community>
</communities>
```

7.2.3. List Communities - unauthenticated

URI format

http://{server}/communitiesanonymous

XML - structure



Path	Description
/community/link	Identification of the Community.
/community/type	Registration type of the Community:‘open’ (everybody can join without approval), ‘closed’ (people can only join after approval of an administrator) or ‘private’ (nobody can join, the community is only accessible for its creator).
/community/registered	As no user is information is used to create the list of communities for the unauthenticated call, the value for this element is always set to “notregistered”.
/community/registered-count	The number of persons that are registered for the Community.

XML - example

```
<communities>
	<community>
		<link type=”community” href=”xxxxxxx”>display title</link>
		<type>open/closed</type>
		<creator>
			<link type=”user” href=”xxxxxxx”>display title</link>
		</creator>
		<registered>notregistered/registered/pending/rejected</registered>
		<registered-count>nnnn</registered-count>
	</community>
</communities>
```

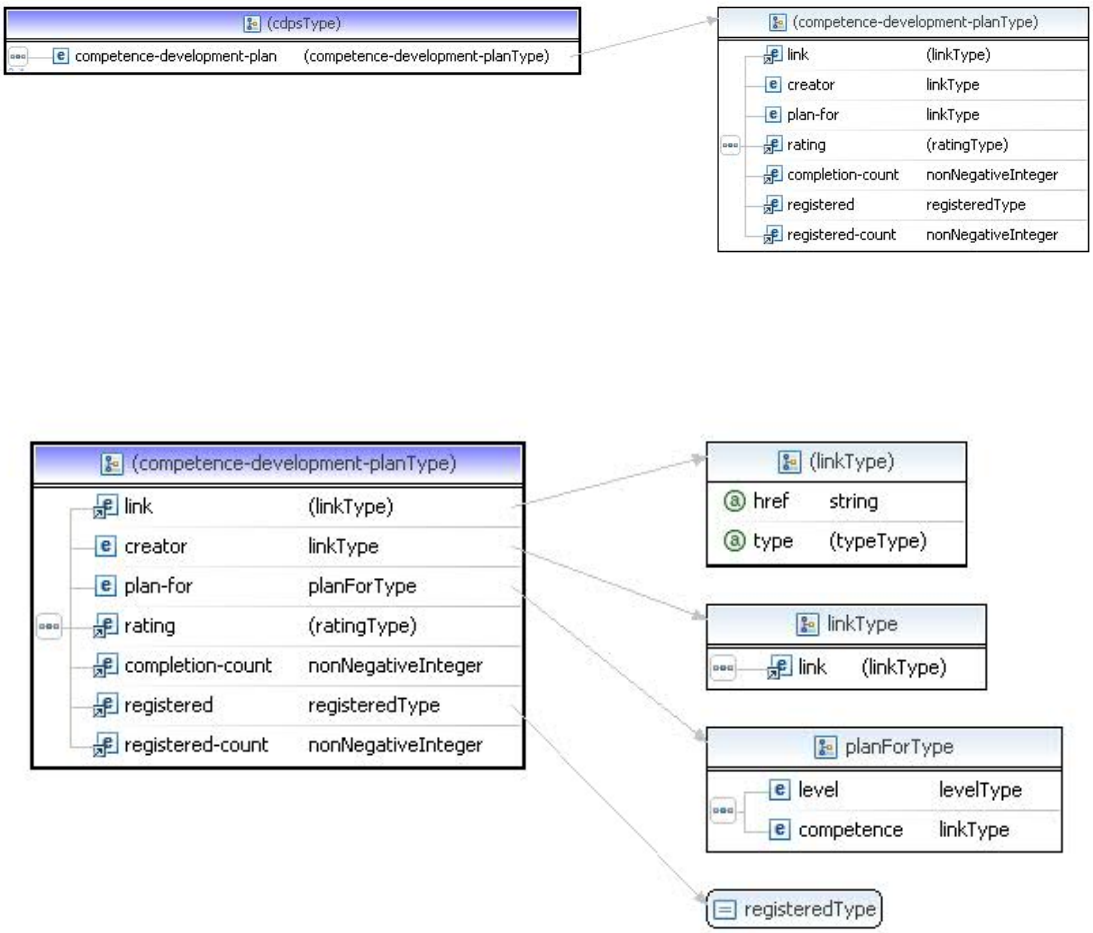
7.2.4. List Competence Development Plans

URI format

http://{server}/{community-guid}.community/cdps?statistics=true&competence=href

Competence parameter is optional

XML - structure



Path	Description
/competence-development-plan/plan-for/level	The competence level for this CDP. When successfully completing the CDP, the user's attained level on the linked Competence is increased to the CDP's level.
/competence-development-plan/plan-for/competence	The Competence to which this plan applies.
/competence-development-plan/rating	Float with range [0..1] indicating the average rating given to this CDP.
/competence-development-plan/completion-count	The number of persons that have completed this CDP.
/competence-development-plan/registered-count	The number of persons that are registered for this CDP.

XML - example

```
<cdps>
	<competence-development-plan>
		<link type="cpd" href="xxxxxxx">display title</link>
		<creator>
			<link type="user" href="xxxxxxx">display title</link>
		</creator>
		<plan-for>
			<level>ddddd</level>
			<competence>
				<link type="competence" href="url">display title</link>
			</competence>
		</plan-for>
		<rating>0.nnn</rating>
		<completion-count>nnn</completion-count>
		<registered>notregistered/registered/pending/rejected</registered>
		<registered-count>nnn</registered-count>
		<rights>
			..
		</rights>
	</competence-development-plan>
</cdps>
```

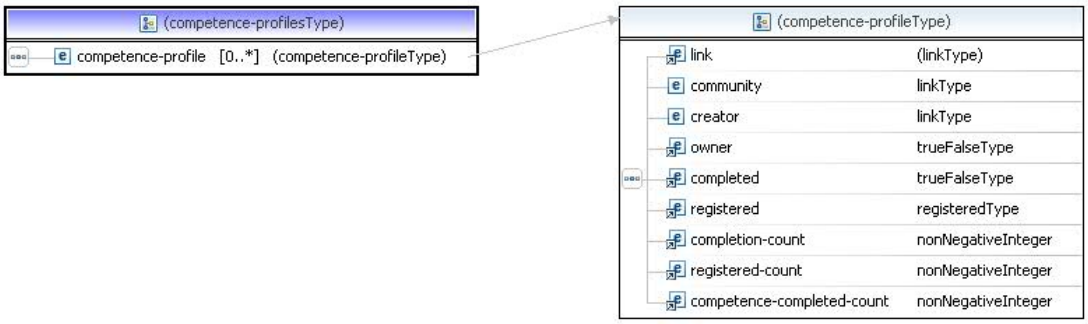
7.2.5. List Competence Profiles

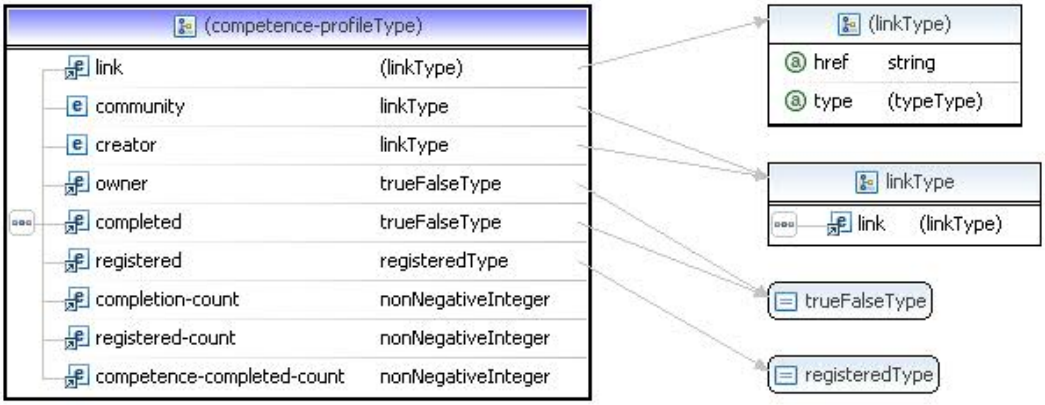
URI format

http://{server}/{community-guid}.community/competence-profiles&statistics=true
http://{server}/competence-profilesanonymous

The call using competence-profilesanonymous returns exactly the same structure, the difference is in which objects are returned. "competence-profiles" returns the accessible profiles of one community to the current user. "competence-profilesanonymous" returns all shared competence profiles from all open and closed communities on the server, regardless of the user.

XML - structure





Path	Description
/competence-profile/link	Identification of the Competence Profile.
/competence-profile/community	Identification of the Community in which the Competence Profile is contained.
/competence-profile/owner	Indicates if the user is the owner of this profile.
/competence-profile/completed	Indicates if the user has completed this Competence Profile: 'true' or 'false'.
/competence-profile/completion-count	The number of persons that have completed the Competence Profile.
/competence-profile/registered-count	The number of persons that have registered for this Competence Profile.
/competence-profile/competence-completed-count	The number of competences referenced in this profile that the user has completed.

XML - example

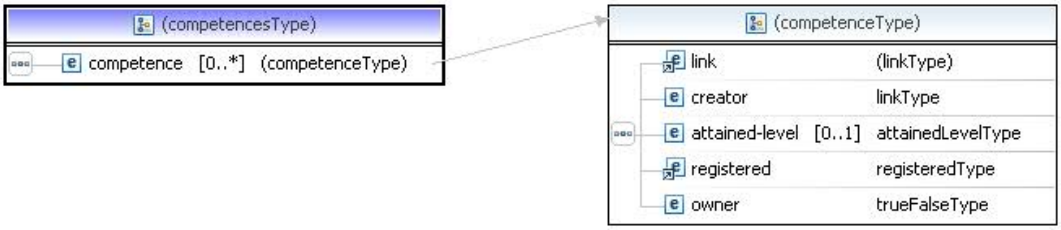
```
<competence-profiles>
	<competence-profile>
		<link type="competence-profile" href="xxxxxxx">display title</link>
		<community>
			<link type="community" href="xxxxxxx">display title</link>
		</community>
		<creator>
			<link type="user" href="xxxxxxx">display title</link>
		</creator>
		<owner>true/false</owner>
		<completed>true/false</completed>
		<registered>notregistered/registered/pending/rejected</registered>
		<completion-count>nnn</completion-count>
		<registered-count>nnnn</registered-count>
		<competence-completed-count>nnn</competence-completed-count>
	</competence-profile>
</competence-profiles>
```

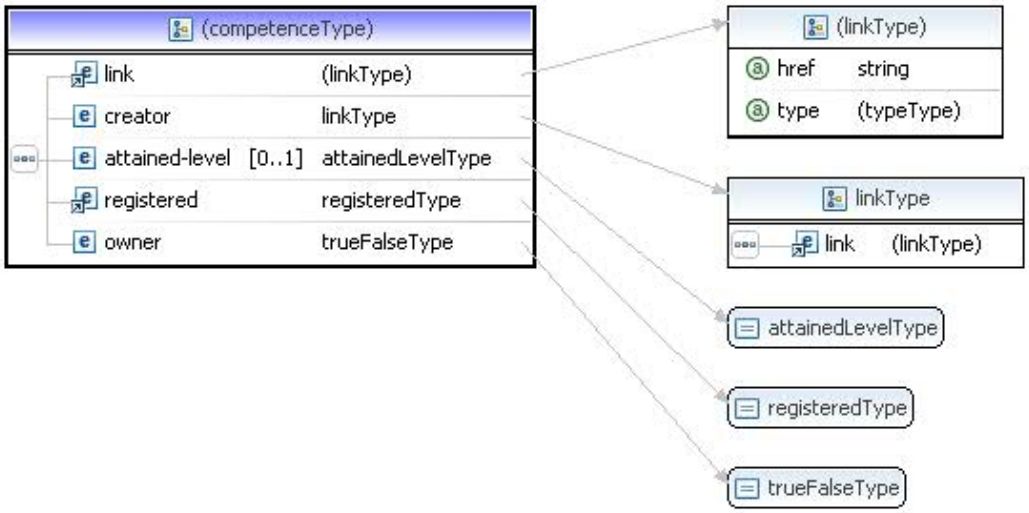
7.2.6. List Competences

URI format

http://{server}/{community-guid}.community/competences

XML - structure





Path	Description
/competence/link	Identification of the Competence.
/competence/attained-level	The current competence level of the user for this Competence. The level is an optional element. If omitted it is not yet known what the level of the user is for this Competence.
/personal/owner	Specifies if the user is the owner of this Competence.

XML - example

```
<competences>
	<competence>
		<link type=”competence” href=”xxxxxxx”>display title</link>
		<creator>
			<link type=”user” href=”xxxxxxx”>display title</link>
		</creator>
		<attained-level>0..8</attained-level>
		<registered>notregistered/registered/pending/rejected</registered>
		<owner>true/false</owner>
	</competence>
</competences>
```

7.2.7. ListItems

URI format

http://{server}/{community-guid}.community/items

XML - structure

Path	Description
/item	Identification of this Assessment Item.

XML - example

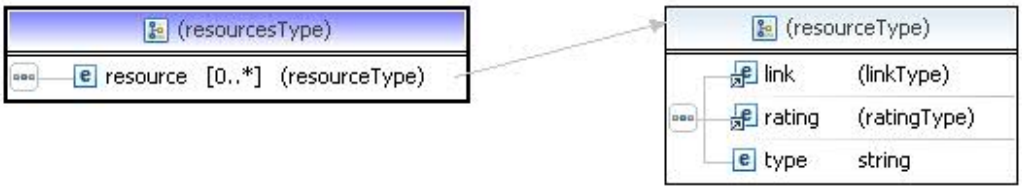
```
<items>
	<item>
		<link type=”item” href=”xxxxxxx”>display title</link>
	</item>
</items>
```

7.2.8. List Resources

URI format

http://{server}/{community-guid}.community/resources

XML - structure



Path	Description
/resource/link	Identification of this Resource.
/resource/rating	Float with range [0..1] indicating the average rating given to this Resource.
/resource/type	'Web resource' (for URI's) or 'File resource' (for resources containing a body).

XML - example

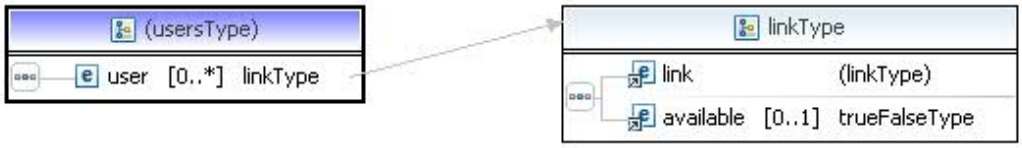
```
<resources>
	<resource>
		<link type="resource" href="xxxxxxx">display title</link>
		<rating>0.nnn</rating>
		<type>type</type>
	</resource>
</resources>
```

7.2.9. List Users

URI format

http://{server}/{community-guid}.community/users
http://{server}/{community-guid}.community/users?availability=true

XML - structure



Path	Description
/user/link	Identification of this User.
/user/available	Indicates if this User is currently on line: 'true' or 'false'. A User is regarded as on line, in case his/her PCM client did a call to the PCM services no longer than 15 minutes earlier.

XML - example

The message below is an example of the XML structure as returned by the server.

```
<users>
	<user>
		<link type="user" href="xxxxxxx">display title</link>
		<available>true></available> *
	</user>
	<user>
		<link type="user" href="xxxxxxx">display title</link>
		<available>true></available> *
	</user>
	<user>
		<link type="user" href="xxxxxxx">display title</link>
		<available>true></available> *
	</user>
</users>
```

* Only returned when parameters 'availability=true' is passed.

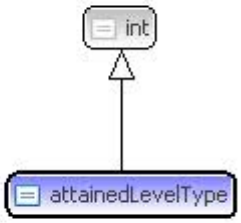
7.3. Common Types

7.3.1. attainedLevelType

Description

The level attained by a user for a certain competence. Expressed as an integer, restricted to range [0..8].

XML - structure

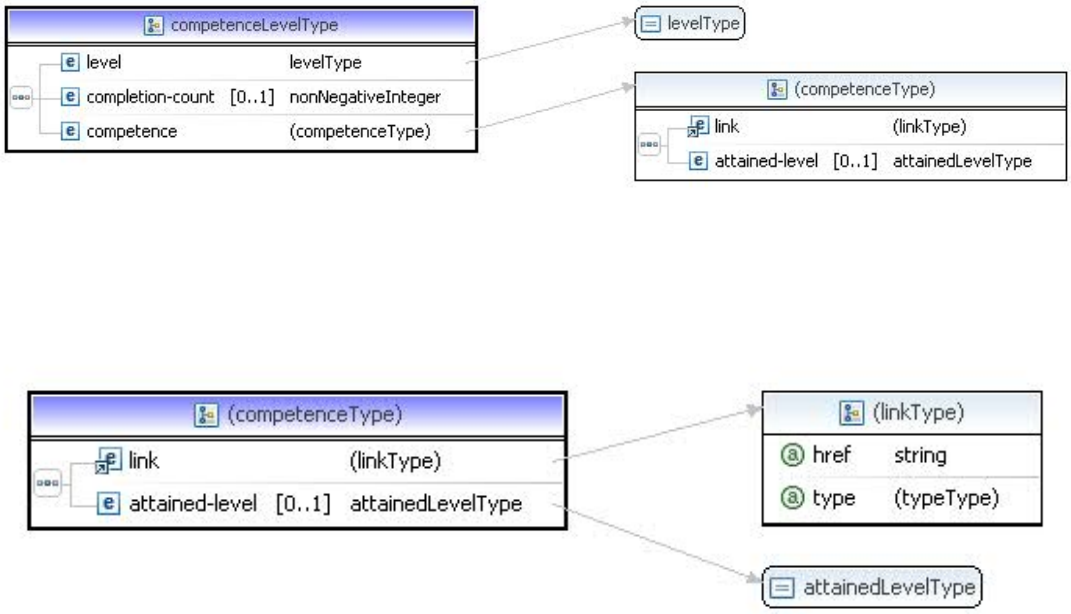


7.3.2. competenceLevelType

Description

The combination of a Competence and a Competence Level.

XML - structure



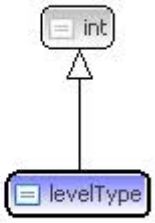
Path	Description
/level	The Competence Level to attain.
/completion-count	The number of persons that have attained this Competence Level or a higher one for this Competence.
/competence	The Competence to attain.
/competence/link	Identification of the Competence to attain.
/competence/attained-level	The Competence Level attained by the current user.

7.3.3. levelType

Description

The competence level for a certain CDP or a Competence in a Competence Profile. Expressed as an integer, restricted to range [0..8].

XML - structure



7.3.4. linkType

Description
Identification for a TENCompetence object.

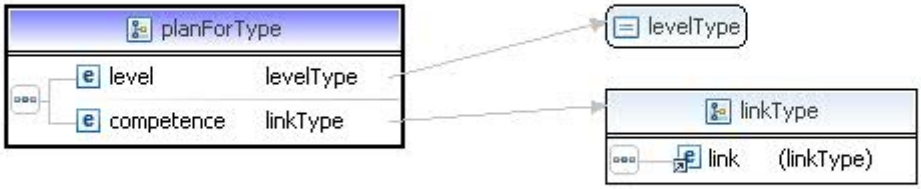
XML - structure

Path	Description
/link, attribute href	The URI of the TENCompetence object.
/link, attribute type	The type of the TENCompetence object (e.g. Competence, Resource, …).

7.3.5. planForType

Description
Indicates the target of a CDP: a Competence and the target Competence Level.

XML - structure



Path	Description
/level	Target Competence Level for the CDP.
/competence	The Competence to attain.

7.3.6. registeredType

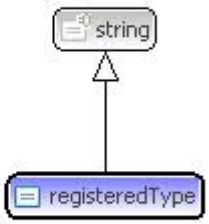
Description

The registration status of the current user for a certain object. Allowed values are:

- 'notregistered' (not registered),
- 'registered' (registered),
- 'pending' (waiting for approval or rejection of the registration request for this closed community) and
- 'rejected' (registration request for this closed community was rejected).

'pending' and 'rejected' currently only occur for Community registration requests. Registration requests for all other objects (CDP's, Competences, …) are automatically approved when the user is a registered member of the enclosing Community.

XML - structure

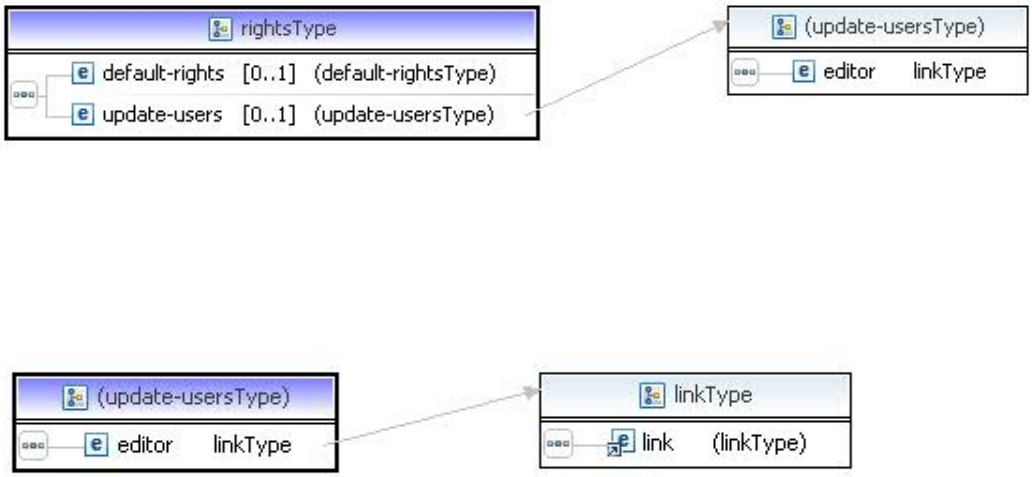


7.3.7. rightsType

Description

The authorisation information for a TENCompetence object, it determines who can read and/or edit the object. In case an object can be deleted, the creator is the only user allowed to do so.

XML - structure



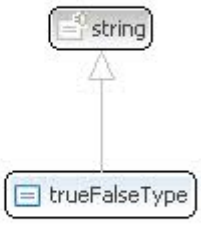
Path	Description
/default-rights	The general authorisation setting for the object. Allowed values are: 'notshared': the object is only visible to the creator. 'shared': the object is readable to everybody in the community, only a named group of users (specified in /update-users) are allowed to edit it. 'sharedforupdate': the object is readable and editable by everybody in the community.
/update-users	Named group of users, who are allowed to edit the object.

7.3.8. trueFalseType

Description

A boolean implementation, allowed values are 'true' and 'false'.

XML - structure



7.4. Auxiliary services

7.4.1. Forum

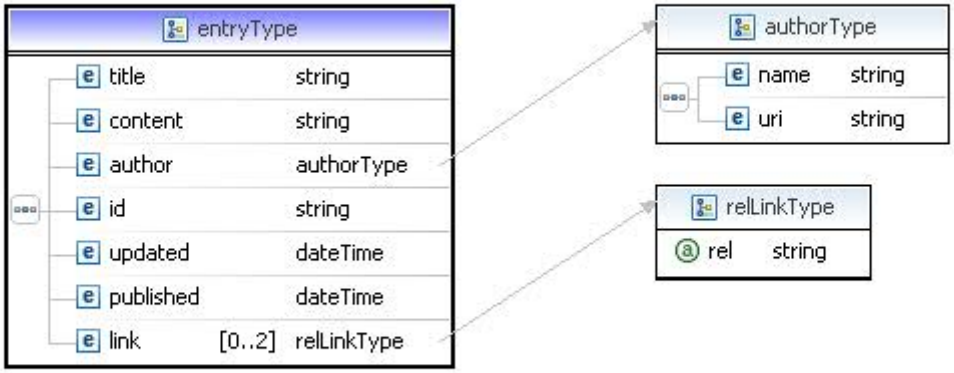
Description

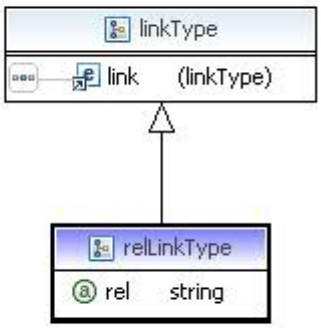
The forum functionality can be used to have asynchronous discussions. A separate 'topic' can be created for each discussion. Each contribution to a discussion is a 'post'.

Services

- Create new topic
  This creates a new topic and automatically creates an initial post for the topic.
- Create new post
- Retrieve all topics for a certain object
  e.g. all topics for a certain Competence or a certain CDP.
- Retrieve a topic, including the details of all its posts
- Retrieve post
- Update topic
- Update post
- Delete topic
- Delete post

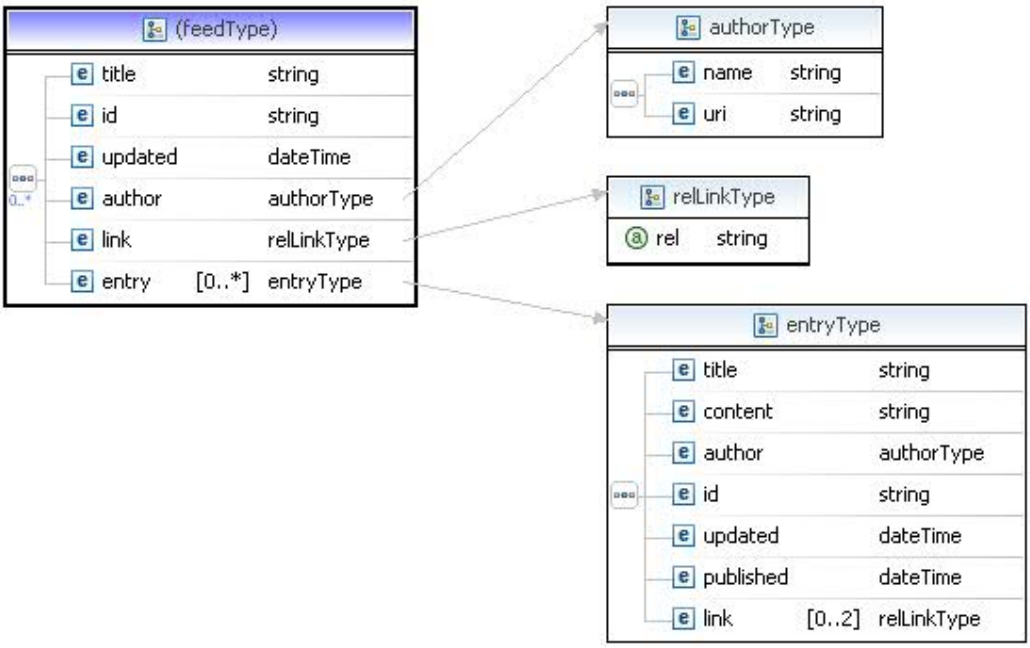
XML – structure for Post

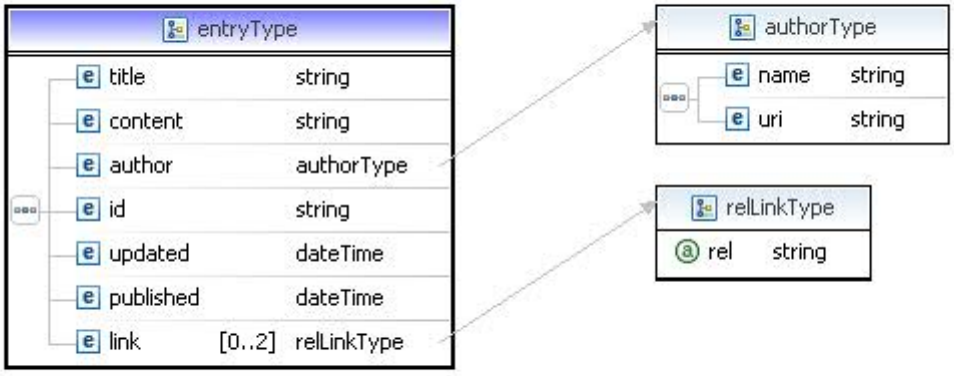


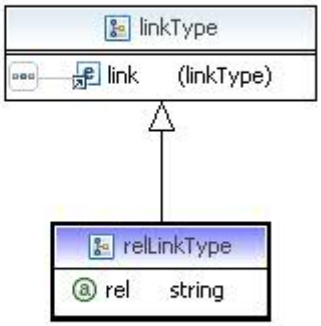


Path	Description
/entry/content	Content (the text) for the Post.
/entry/author/name	Display name of the author.
/entry/author/uri	URI for the author of the Post.
/entry/id	Identification of the Post, in the form of a URI (instead of a “linkType” element).
/entry/updated	Date and time on which the Post was updated last.
/entry/published	The date and time on which this Post was originally created.
/entry/link, for attribute rel=”self”	Link to the post itself.
/entry/link, for attribute rel=”up”	Link to the parent of this post (the topic itself or a previous post).

XML –structure for Topic







Path	Description
/entry/id	Identification of the Topic, in the form of a URI (instead of a “linkType” element).
/entry/updated	Date and time on which the Topic was created. Note: this is different than the element name is expressing!
/entry/author/name	Display name of the author.
/entry/author/uri	URI for the author of the Topic.
/entry/link, attribute rel=”self”	Link to the topic itself.
/entry/…	See the table above for Post.

Note: the messages currently also contain the deprecated element /feed/entry/category. This element will be removed in the near future and should not be used.

XML –structure for List topics

The structure for List Topics is very similar to the structure of GET Topic. List Topics omits the fields from the following list, for the rest it's the same:

- /feed/updated
- /feed/author
- /feed/entry/content
- /feed/entry/published
- /feed/entry/link, for attribute rel="self"

XML - allowed elements and example for Post

```
CRU <entry>
CRU    <title>post2</title>
CRU    <content>my2 content</content>
 R     <author>
 R            <name>Pete Richards</name>
 R            <uri>http://localhost/TENCServer/user/457fbea.user</uri>
       </author>
 R
       <id>http://localhost/TENCServer/f3f75f0b.community/topic1.topic/post2.post</i
d>
 R     <updated>2007-05-15T10:16:35.140+02:00</updated>
 R     <published>2007-05-15T10:16:35.140+02:00</published>
 R     <link type="text"

       href="http://localhost/TENCServer/f3f75f0b.community/topic1.topic/post2.post"
              rel="self" />
CR     <link type="text"
              href="http://localhost/TENCServer/f3f75f0b.community/topic1.topic"
rel="up" />
</entry>
```

XML - allowed elements and example for Topic

Note: the <feed> and <title> elements are the only ones allowed for an update PUT. All other elements are only allowed for a read (GET).

```
<feed xmlns="http://www.tencompetence.org/api/v1.0">
       <title>Topic 1</title>
       <id>http://localhost:8080/TENCServer/...topic</id>
       <updated>2007-11-28T17:47:10.000+01:00</updated>
       <author>
              <name>Ruud Lemmers</name>
              <uri>http://localhost:8080/TENCServer/user/ruud.user</uri>
       </author>
       <link type="application/atom+xml"
                           href="http://localhost:8080/TENCServer/...topic"
rel="self" />
       <entry>
              <title>Topic 1</title>
              <content>Some text for the first post within Topic 1.</content>
              <author>
                     <name>Ruud Lemmers</name>
                     <uri>http://localhost:8080/TENCServer/user/ruud.user</uri>
              </author>
              <id>http://localhost:8080/TENCServer/..post</id>
              <updated>2007-11-28T17:47:10.000+01:00</updated>
              <published>2007-11-28T17:47:10.000+01:00</published>
```

```
		<link type="text" href="http://localhost:8080/TENCServer/...post"
rel="self" />
		<link type="application/atom+xml"
				href="http://localhost:8080/TENCServer/...topic" rel="up"
/>
	</entry>
	<entry>
		<title>Re: Topic 1</title>
		<content>>>> Ruud Lemmers wrote: >> Some text for the first post within
Topic 1. And this is my reply!</content>
		<author>
			<name>Ruud Lemmers</name>
			<uri>http://localhost:8080/TENCServer/user/ruud.user</uri>
		</author>
		<id>http://localhost:8080/TENCServer/...post</id>
		<updated>2007-11-28T17:47:39.000+01:00</updated>
		<published>2007-11-28T17:47:39.000+01:00</published>
		<link type="text" href="http://localhost:8080/TENCServer/...post"
rel="self" />
		<link type="text" href="http://localhost:8080/TENCServer/...post"
rel="up" />
		<category term="6f49eb67-1183-4144-ac1e-66c26ab1ede3" />
	</entry>
</feed>
```

XML - allowed elements and example for List Topics

```
<feed xmlns="http://www.tencompetence.org/api/v1.0">
	<title>Community 2</title>
	<updated>2007-11-28T17:48:41.196+01:00</updated>
	<id>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community</id>
	<link type="community" href="http://localhost:8080/TENCServer/549ad2df-ef7d-
4bc1-9ff9-182f9caf3a57.community" rel="self">Community 2</link>
	<entry>
		<author>
			<name>Ruud P. Rogrammer </name>
			<uri>http://localhost:8080/TENCServer/user/ruud.user</uri>
		</author>
		<title>Topic 1</title>
		<id>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/08f9ae79-1bbc-4442-8fb5-eb39cff9b911.topic</id>
		<updated>2007-11-28T17:47:10.000+01:00</updated>
		<link type="application/atom+xml"
href="http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/08f9ae79-1bbc-4442-8fb5-eb39cff9b911.topic" rel="self" />
	</entry>
	<entry>
		<author>
			<name>Ruud P. Rogrammer</name>
			<uri>http://localhost:8080/TENCServer/user/ruud.user</uri>
		</author>
		<title>Topic 2</title>
		<id>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/b95c4e37-4624-4aea-a074-dae4547db9a7.topic</id>
		<updated>2007-11-28T17:47:54.000+01:00</updated>
		<link type="application/atom+xml"
href="http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/b95c4e37-4624-4aea-a074-dae4547db9a7.topic" rel="self" />
	</entry>
</feed>
```

7.4.2. Message

Description

Services

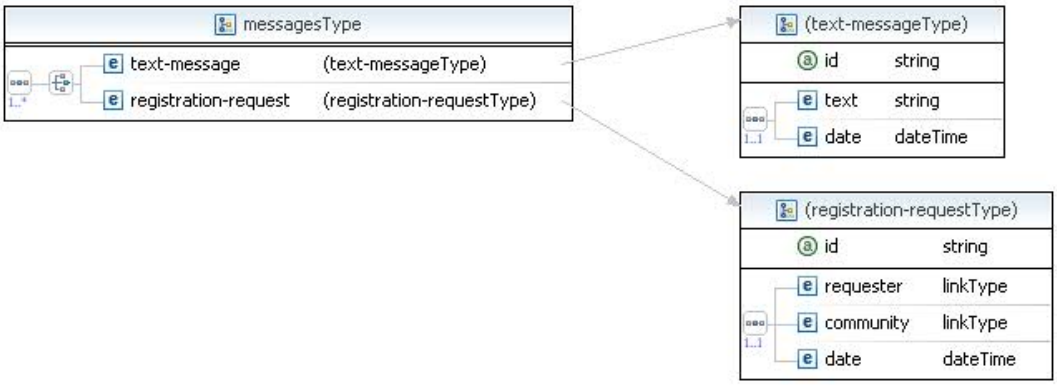
- Retrieve the messages sent to a specific community
  Specify the URL of the community, for which the messages should be retrieved as parameter 'recipient'.
- Retrieve the messages sent to a specific user
  Don't specify the 'recipient' parameter, retrieving messages for the current user is the default.

URI format

http://{server}/messages?recipient=<url of recipient, which has to be a URL to a community>
http://{server}/messages?recipient=<url of recipient, which has to be a URL to a community>&offset=<a positive integer value>

Both recipient and offset are optional parameters.

XML - structure



Path	Description
/text-message/text	Text for a message.
/text-message/date	Date and time on which the message was generated.
/registration-request/requester	Identification of the user that requested access to a community.
/registration-request/community	Identification of the community to which the user requested access.
/registration-request/date	Date and time on which the user requested access.

XML - allowed elements and example

```
<messages xmlns="http://www.tencompetence.org/api/v1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.tencompetence.org/api/v1.0
http://www.tencompetence.org/api/v1.0">
       <text-message id="12">
              <text>Hubert Vogten has been added to a community</text>
              <date>2007-03-15T19:30:44.000+02:00</date>
       </text-message>
       <registration-request id="17">
              <requester>
                     <link type="user" href="http://../harrie.user">Harrie</link>
              </requester>
              <community>
                     <link type="community" href="http://../guid1.community">Java
Boyz</link>
              </community>
              <date>2007-04-19T16:00:31.000+02:00</date>
       </registration-request>
       <text-message id="14">
              <text>Your registration request for Java Boyz has been denied</text>
              <date>2007-06-19T08:49:41.000+02:00</date>
       </text-message>
       <text-message id="15">
              <text>Welcome to Java Boyz</text>
              <date>2007-08-22T08:23:58.000+02:00</date>
       </text-message>
</messages>
```

7.4.3. Rating

Description

Used to rate the objects within the PCM. Users can score an object anywhere between 0 and 1 and can provide an optional comment on their rating, to express their opinion about the object's quality.

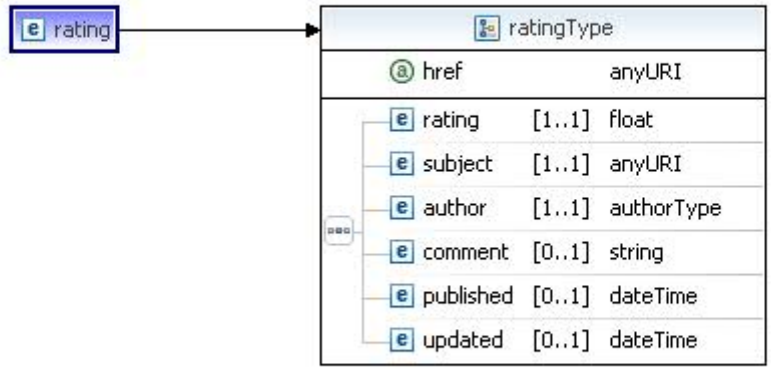
Services

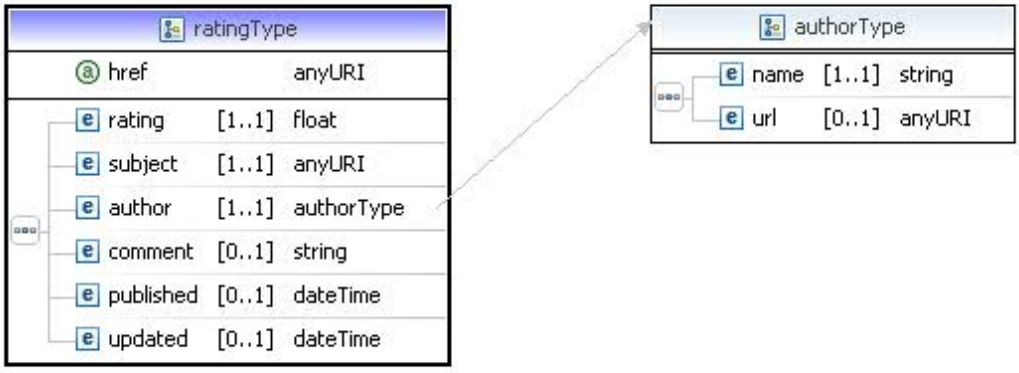
- Create
- Retrieve
- Retrieve list: summary info for all ratings from a specific object (called 'subject' in the parameter).

URI format

POST a rating: http://{server}/rating
GET an existing rating: http://{server}/rating/item/{number}

XML – structure





Path	Description
Attribute href	The identification assigned by the server to a new rating object.
/rating	Float with range [0..1] indicating the rating value given.
/subject	Identification of the object that was rated.
/author/name	Display name of the author.
/author/url	Identification of the author.
/comment	Textual comment explaining the rating.
/published	Date and time on which the rating object was initially published
/updated	Date and time on which the rating object was updated last.

XML – allowed elements and example

All elements shown below are allowed for both the POST and GET operation.

```
<?xml version="1.0" encoding="UTF-8"?>
	<rating xmlns="http://www.tencompetence.org/rating/">
	<rating>0.9</rating>
	<subject>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/c53fd134-469b-47e2-978d-5e455e30d87c.competence-development-
plan</subject>
	<author>
		<name>Ruud P. Rogrammer</name>
		<url>http://localhost:8080/TENCServer/user/ruud.user</url>
	</author>
	<comment>Very good CDP.</comment>
</rating>
```

7.4.4. List Ratings

Description

Returns all rating objects that conform to the specified filter parameters.

URI format

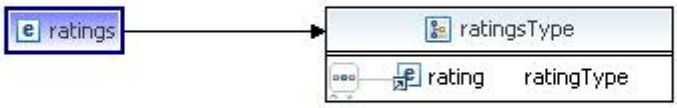
http://{server}/rating?subject=...&author=...&format=...

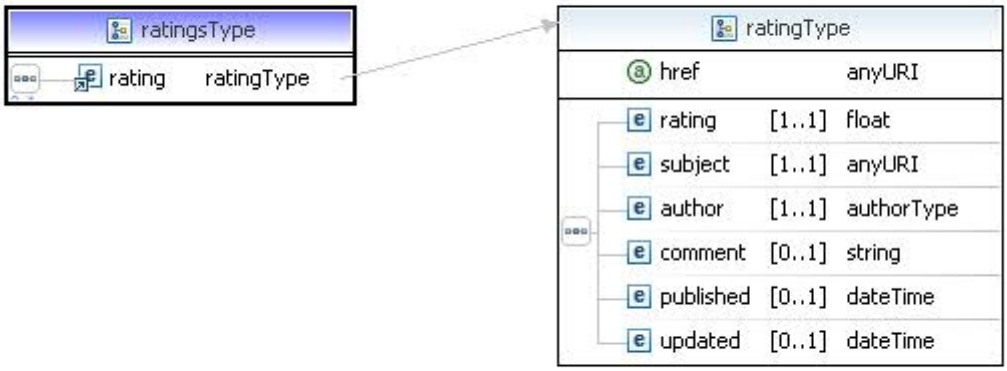
Parameters:

- subject: retrieve only ratings for this subject (URI of a TENCompetence object).
- author: retrieve only ratings of this author (author URI).
- format: retrieve ratings in a summary format.

Note: it's allowed to use multiple parameters. Use of both 'subject' and 'author' will return only rating objects that have that 'subject' AND 'author'.

XML - structure





Path	Description
/rating	A rating object.
Attribute href	The identification assigned by the server to a new rating object.
/rating	Float with range [0..1] indicating the rating value given.
/subject	Identification of the object that was rated.
/author/name	Display name of the author.
/author/url	Identification of the author.
/comment	Textual comment explaining the rating.
/published	Date and time on which the rating object was initially published
/updated	Date and time on which the rating object was updated last.

XML – allowed elements and example

As this applies only to one call (GET), all elements shown in the example are allowed.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rat:ratings xmlns:rat="http://www.tencompetence.org/rating/">
	<rat:rating href="http://localhost:8080/TENCServer/rating/5.item">
		<rating>0.7</rating>
		<subject>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/c53fd134-469b-47e2-978d-5e455e30d87c.competence-development-
plan</subject>
		<author>
			<name>Ruud Lemmers</name>
			<url>http://localhost:8080/TENCServer/user/ruud.user</url>
		</author>
		<comment>Reasonable object.</comment>
		<published>2007-11-26T15:26:57.000+01:00</published>
		<updated>2007-11-26T15:26:57.000+01:00</updated>
	</rat:rating>
	<rat:rating href="http://localhost:8080/TENCServer/rating/4.item">
		<rating>0.9</rating>
		<subject>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/c53fd134-469b-47e2-978d-5e455e30d87c.competence-development-
plan</subject>
		<author>
			<name>Ruud Lemmers</name>
			<url>http://localhost:8080/TENCServer/user/ruud.user</url>
		</author>
		<comment>Very good CDP.</comment>
		<published>2007-11-26T15:23:02.000+01:00</published>
		<updated>2007-11-26T15:23:02.000+01:00</updated>
	</rat:rating>
	<rat:rating href="http://localhost:8080/TENCServer/rating/3.item">
		<rating>0.5</rating>
		<subject>http://localhost:8080/TENCServer/549ad2df-ef7d-4bc1-9ff9-
182f9caf3a57.community/c53fd134-469b-47e2-978d-5e455e30d87c.competence-development-
plan</subject>
		<author>
			<name>Ruud Lemmers</name>
			<url>http://localhost:8080/TENCServer/user/ruud.user</url>
		</author>
		<comment/>
		<published>2007-11-23T15:59:39.000+01:00</published>
		<updated>2007-11-23T15:59:39.000+01:00</updated>
	</rat:rating>
</rat:ratings>
```

7.4.5. Server Discovery (repository)

Description

The Discovery service is the repository for the URI's of all known TENCompetence servers. After setting up a TENCompetence server, it should be registered in the repository.

Services

- Add a server
  Adds a server to the repository. As this is an update to the list of servers, it is done by using a PUT (update).
- Retrieve a list of all known TENCompetence servers.

URI format

http://{server}/ servers

XML – structure for List Servers (GET)

XML – structure for Add Server (PUT)

XML – allowed elements and example for List Servers (GET)

All elements in the example are allowed and mandatory.

```
<servers>
	<server>
		<link type="server"
		href="http://localhost:8080/TENCServer">My local server</link>
	</server>
	<server>
		<link type="server"
		href="http://tenc.somewhere.nl/TENCServer">The Dutch Server</link>
	</server>
	<server>
		<link type="server"
		href="http://www.tencompetence.org/TENCServer">Default Server</link>
	</server>
</servers>
```

XML – allowed elements and example for Add Server (PUT)

All elements in the example are allowed and mandatory.

```
<server>
	<link type="server"
		href="http://tenc.somewhere.nl/TENCServer">The Dutch Server</link>
</server>
```

Path	Description
/server	The information regarding one TENCompetence server.
/server/link	Identification of the TENCompetence server.

7.4.6. Other - Chat

This functionality has been implemented by integrating an external chat server product (OpenFire). It's not addressable as a service.

8. Authorisation rules

8.1. Introduction

When looking at what a user wants/needs to do related to authorisation, we can distinguish the following list:

- Supply object to others for use
- Collaborate on object (to edit/compose an object with multiple persons)
- Grant/Revoke write access to/from an object from others, by the author.
- Reuse existing Action from others to build a CDP (or Resource to build an Action)
- Grant someone access to join a community
- Grant/Revoke the granting right (which lets someone accept/reject access to a closed community), by the author.
- Create a private copy of an object

Authorisation can be set for four different object types:

1. Communities
2. Competence Profiles
3. Competence Development Plans
4. Actions

Authorisation rights for Resources and Competences are derived from the settings for the other object types.

The remaining sections in this appendix describe what rules are implemented for the authorisation rights and what the exact effects of authorisation settings are to users. The authorisation rules are implemented in both the PCM client and the PCM server.

8.2. GUI for Competence Profile, CDP and Action authorisation

The authorisation options for Competence Profiles, CDPs and Actions are set in a very similar way. The table below shows the different options and their general effect.

Option	Object visible to	Object changeable by
Do not share	Creator only	Creator only
Share; everyone in the community can change	Everybody in the community, to which the object belongs.	Creator and everybody in the community, to which the object belongs.
Share; only I and the following persons can change	Everybody in the community, to which the object belongs.	Creator and a list of named persons.

Rights (grants) can only be updated by the owner.

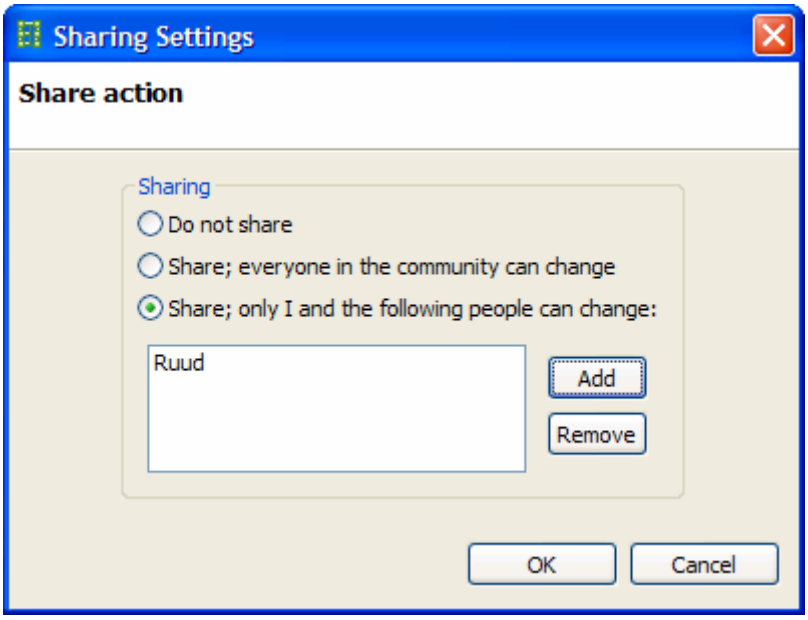


Figure 12 User Interface for setting authorisation level

Break synchronisation

Users can select ”Break synchronization” in the GUI for Competence Profiles, CDPs and Actions to create a new private copy for the logged on user. This functionality is available for the objects to which the user has read rights (and thereby for all visible objects). The object created has the “Do not share” rights like all other newly created objects.

Breaking synchronisation only creates shallow copies. Contained objects are not copied, references to the already existing objects will be used.
Example: breaking synchronization for a CDP will create references to the actions of the original CDP. Thus, it will not copy the contained actions from the original CDP to refer to those.

8.3. GUI for Community authorisation

The authorisation options for communities are different than for the other types of objects. The table below shows the options and their general effect.

Option	Community visible to	Addition of members to the community	Community changeable by
Private community	Creator only	N/A (other people can't register)	Creator only
Open community	Everyone	N/A (people can register themselves)	All community members
Closed community and also the following persons may add members	Everyone	Creator and a list of named persons can accept or decline access requests (these access requests are shown in the Agent view)	All community members

Rights (grants) can only be updated by the owner. The owner acts as administrator in this case.

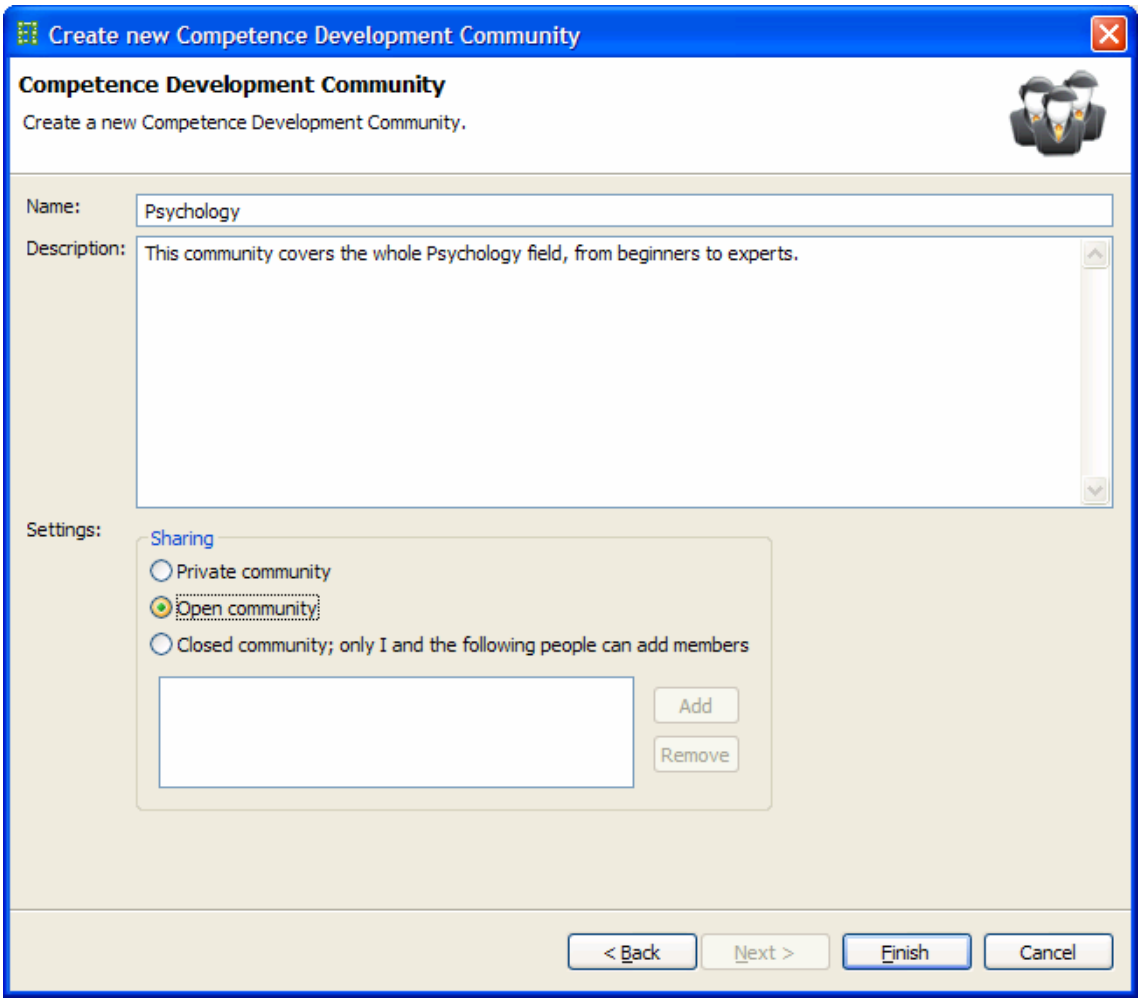


Figure 13 User Interface for creating CDP

8.4. Additional rules

- Authorisation can NOT be set separately for a Competence or a Resource.
- Sharing a CDP is only allowed when all its Actions are already shared.
- Decreasing the accessibility from shared to not shared is never allowed.
- Reducing the visibility for a community from open or closed to private is never allowed.
- Rights (grants) can only be updated by the owner of an object.
- Only the owner (= the creator) of an object is allowed to delete it.

8.5. Allowed functionality

This section shows in detail how the user's functionality is limited by the settings done in the GUI.

Notes:
1. The term "sharedForUpdate" is used as shorthand for "'Share and everyone in the community can change' or 'Share and the following persons can change, besides me - where the logged on user is part of the named persons'".
2. The delete function isn't implemented in the PCM client. Because of this, the functionality has been tested less thoroughly than other functionality. There can be bugs in it.

Action
Create Action: always allowed.
Read Action: allowed if the Action is owned/shared.
Update Action: allowed if the Action is owned/sharedForUpdate.
Delete Action: allowed if the Action is owned and not part of any CDP.
Precondition for Create/Read/Update: user is member of the encapsulating community.

Resource
Create Resource: always allowed.
Read Resource: always allowed.
Update Resource: allowed if the Resource is owned.
Delete Resource: allowed if the Resource is owned and not part of any Action.
Precondition for Create/Read/Update: user is member of the encapsulating community.

CDP
Create CDP: always allowed.
Read CDP: allowed if the CDP is owned/shared.
Update CDP: allowed if the CDP is owned/sharedForUpdate.
Delete CDP: allowed for the owner, when there's no registered user for it.
Precondition for Create/Read/Update: user is member of the encapsulating community.

Community
Create Community: always allowed.
Read Community: always allowed for open/closed communities, allowed only to the owner for private communities.
Update Community: allowed if member of the community.
Delete Community: allowed for owner, when there are no other users in the community.
Precondition: -

Competence
Create Competence: always allowed.
Read Competence: always allowed.
Update Competence: allowed for owner.
Delete Competence: allowed for owner, when there's no CDP for the Competence and there are no Competence Profiles containing the Competence.
Precondition for Create/Read/Update: user is member of the encapsulating community.

User
Create User: always allowed
Read User: every user on a certain server can read the profiles of all other users on that server.
Update User: users are only allowed to update their own user profile (don't confuse this with competence profiles).
Delete User: users are only allowed to delete their own user profile (don't confuse this with competence profiles).
Precondition: logged in user is member of the same server.

Competence Profile
Create CompetenceProfile: always allowed.
Read CompetenceProfile: allowed if the Comp. Profile is owned/shared.
Update CompetenceProfile: allowed if the Comp. Profile is owned/sharedForUpdate.
Delete CompetenceProfile: allowed for owner, when there are no CDP's for the Comp. Profile.
Precondition for Create/Read/Update: user is member of the encapsulating community.

Lists
1. ListUsers for a community: allowed if member of the community.
2. ListCompetenceDevelopmentPlans for a community: allowed if member of the community.
   Filtered out: unshared CDPs for which the user is not the owner are filtered out.
3. ListActions for a community: allowed if member of the community.
   Filtered out: unshared Actions for which the user is not the owner are filtered out.
4. ListResources for a community: allowed if member of the community.
5. ListCompetenceProfiles for a community: allowed if member of the community.
   Filtered out: unshared Competence Profiles for which the user is not the owner.
6. ListCompetences for a community: allowed if member of the community.
7. ListCommunities for a server: allowed if member of the community.
   Filtered out: private Communities for which the user is not the owner.
8. ListCommunitiesAnonymous for a server: always allowed.
   Filtered out: private Communities for which the user is not the owner.
9. DiscoverServers: always allowed.

9. Recommended introductory reading

This document assumes readers are familiar with the TENCompetence project and its concepts. People who are completely new to TENCompetence are recommended to study the following documents:

1. Briefing about the PCM ([24])
   This briefing explains the TENCompetence vision and clarifies what a Personal Competence Manager is / does.
2. Domain model ([2])
   This document defines the logical domain model which underlies all project development work. It shows the model in a class diagram, it has extensive descriptions (including examples) of the classes and it introduces a set of six use cases. The domain model document strongly influenced the WP3 work.
3. Status of the project and description of usage profiles ([25])
   This document describes the status of the project after the first two years. It also describes how the project will proceed, by defining a set of “usage profiles”. Each usage profile will result in a separate client application.

10. References

[1] Kruchten, P. (1995), *The 4+1 View Model of Architecture*, http://doi.ieeecomputersociety.org/10.1109/52.469759

[2] Koper, R. (07-06-2006), *TENCompetence Domain Model*, http://dspace.ou.nl/handle/1820/649.
Note: this is an annex of [3]

[3] Arjona, M., Sánchez, A., Sacristán, N., Glahn, C., Specht, M. and Verjans, S. (2007), *D2.1 Integrated Roadmap*, http://hdl.handle.net/1820/1142.

[4] *Eclipse – an open development platform*, http://www.eclipse.org/.

[5] *The IMS Learning Design Engine*, http://coppercore.org/.

[6] Lemmers, R., Beauvoir, P. (2007), *D3.1 Architecture Design Document*, http://dspace.ou.nl/handle/1820/882.

[7] Arenales, J., Beauvoir, P., Buskens, J., Celle, R., Cherian, R., Dicerto, M., Fishchuck, A., Heyenrath, S., Koesling, A., Lemmers, R., Loizos, M., Martens, H., Mendez, C., Sharples, P., Vogten, H., Wilson, S. (2007), *D3.2: First tested version of the Integrated TENCompetence System*, http://hdl.handle.net/1820/1004.

[8] Arjona, M., Lemmers, R., Glahn, C., Sacristán, N. (2007), *ID2.5 Elaborated Use Cases & Domain Model*, http://hdl.handle.net/1820/1145.

[9] *SourceForge Home Page*, http://sourceforge.net.

[10] *TENCompetence SourceForge*, http://sourceforge.net/projects/tencompetence.

[11] *Wush TENCompetence Bugzilla*, http://wush.net/bugzilla/rulem/.

[12] *Bugzilla*, http://www.bugzilla.org.

[13] *JUnit Home Page*, http://www.junit.org.

[14] *JUnit Tutorial*, http://clarkware.com/articles/JUnitPrimer.html

[15] *The New Features of JUnit 4*, http://www.devx.com/Java/Article/31983.

[16] Massol, V. with Husted, T. (2004), *JUnit in Action*, Manning Publications Co. USA.

[17] *CVS Home Page*, http://www.nongnu.org/cvs/.

[18] *SourceForge and CVS*, http://sourceforge.net/docs/E04/.

[19] Fogel, Bar, *Open Source Development with CVS 2nd Edition*, http://cvsbook.red-bean.com/cvsbook.html.

[20] *Eclipse*, http://www.eclipse.org.

[21] *Eclipse User Interface Guidelines*, http://wiki.eclipse.org/index.php/User_Interface_Guidelines.

[22] Sun Microsystems (1997), *Java Code Conventions*, http://java.sun.com/docs/codeconv/CodeConventions.pdf.

[23] *Hibernate*, http://www.hibernate.org/.

[24] Kew, C., *The TENCompetence Personal Competence Manager*, http://www.tencompetence.org/node/96

[25] Koper, R., Schoonenboom, J., Manderveld, J., Kluijfhout, E., Arjona, M., Griffiths, D., Angehrn, A., Van Rosmalen, P., *D2.2 Updated use case models and underlying vision documents and pedagogical model definitions*, http://hdl.handle.net/1820/1152.

Links checked on 21-02-2008.